SYNCHRONOUS FAILURE DIAGNOSIS OF DISCRETE-EVENT SYSTEMS

Felipe Gomes de Oliveira Cabral

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia Elétrica.

Orientador: Marcos Vicente de Brito Moreira

Rio de Janeiro
Outubro de 2017

SYNCHRONOUS FAILURE DIAGNOSIS OF DISCRETE-EVENT SYSTEMS

Felipe Gomes de Oliveira Cabral

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

____

Prof. João Carlos dos Santos Basilio, Ph.D.

____

Prof. Marcos Vicente de Brito Moreira, D.Sc.

____

Prof. José Eduardo Ribeiro Cury, Docteur d'Etat

____

Prof. Max Hering de Queiroz, D.Sc.

____

Prof. Antonio Eduardo Carrilho da Cunha, D.Eng.

RIO DE JANEIRO, RJ – BRASIL
OUTUBRO DE 2017

Cabral, Felipe Gomes de Oliveira

Synchronous failure diagnosis of discrete-event systems/Felipe Gomes de Oliveira Cabral. – Rio de Janeiro: UFRJ/COPPE, 2017.

XVI, 127 p.: il.; 29, 7cm.

Orientador: Marcos Vicente de Brito Moreira

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2017.

Referências Bibliográficas: p. 119 – 127.

1. Failure diagnosis. 2. Synchronous diagnosis. 3. Discrete-event systems. I. Moreira, Marcos Vicente de Brito. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Porque dele, e por meio dele, e para ele são todas as coisas. A ele, pois, a glória eternamente. Amém! (Romanos 11. 36)

Agradecimentos

Agradeço a Deus, criador e sustentador de todas as coisas, que pela sua graça me concedeu essa conquista.

Agradeço aos meus pais Ronaldo e Denise por sempre acreditarem em mim. Sempre pude e posso contar com vocês.

Agradeço à minha esposa Julia pelo companheirismo e paciência durante todo o período de doutorado e em especial no último ano, que se provou ser o mais difícil.

Agradeço a todos os irmãos em Cristo que congregam na Igreja Batista no Horto, Igreja Evangélica Congregacional de Higienópolis (IECH) e Primeira Igreja Batista no Andaraí (PIBA) por todos os momentos de comunhão comigo e com minha esposa.

Agradeço ao meu orientador e amigo, Marcos Moreira, por todas as horas de aconselhamento e orientação que não se limitam ao escopo deste trabalho.

Agradeço a todos os amigos e companheiros que fiz ao longo da graduação. Em especial, aos grandes amigos do curso de Engenharia Elétrica da UFRJ: Rafael Mazza, Mayara Cagido, Rafael Caetano, Tiago Granato e Victor Portavales. O companheirismo de vocês foi, e continua, sendo fundamental.

Agradeço a todos os amigos que fiz no Laboratório de Controle e Automação (LCA) da COPPE/UFRJ que me acompanharam durante o mestrado e doutorado. Em especial, Wesley Silveira, Públio Lima, Tiago França, Juliano Freire, Jean Tomola, Félix Gamarra, Thiago Henrique, Gustavo Viana e Marcos Vinicius.

Agradeço a todos os alunos e amigos que fiz durante o período em que fui professor substituto do Departamento de Engenharia Elétrica da UFRJ. Honestamente, posso dizer que aprendi mais com vocês do que ensinei.

Agradeço a todos os professores da COPPE/UFRJ que de forma direta ou indireta contribuíram com a minha formação.

Agradeço ao Conselho Nacional de Desenvolvimento Tecnológico e Científico (CNPq) pelo suporte financeiro.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

DIAGNÓSTICO SÍNCRONO DE FALHAS DE SISTEMAS A EVENTOS DISCRETOS

Felipe Gomes de Oliveira Cabral

Outubro/2017

Orientador: Marcos Vicente de Brito Moreira

Programa: Engenharia Elétrica

Em geral, sistemas são formados pela composição de diversos módulos, componentes locais ou subsistemas e podem ter um grande número de estados. O crescimento do modelo global do sistema com o número de componentes leva a altos custos computacionais para técnicas de diagnóstico de falhas baseadas no modelo global da planta. Neste trabalho, uma nova abordagem para o diagnóstico de falhas de sistemas a eventos discretos é proposta. O método é baseado no cálculo de um diagnosticador rede de Petri, chamado de diagnosticador rede de Petri sincronizado que é construído a partir do comportamento sem falha dos módulos do sistema. A definição de diagnosticabilidade síncrona da linguagem de um sistema em relação à linguagem de seus módulos, e um algoritmo para verificar essa propriedade também são propostos. Uma generalização do diagnosticador síncrono para uma arquitetura decentralizada, a noção de codiagnosticabilidade síncrona e um algoritmo para verificar essa propriedade também são apresentados neste trabalho. A eficiência do diagnóstico síncrono pode ser melhorada usando o modelo sem falha do sistema global, o que leva à definição de diagnosticabilidade síncrona condicional. Um algoritmo para a verificação da diagnosticabilidade síncrona condicional baseado no método de verificação da diagnosticabilidade síncrona é proposto. A relação entre diagnosticabilidade, diagnosticabilidade síncrona, diagnosticabilidade síncrona condicional e codiagnosticabilidade síncrona de sistemas a eventos discretos também é discutida. Algoritmos para o cálculo do atraso máximo para todos os métodos de diagnóstico apresentados neste trabalho são propostos. Um exemplo teórico e uma implementação prática dos métodos de diagnóstico são apresentados e usados ao longo deste trabalho com o objetivo de ilustrar e validar os métodos.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

SYNCHRONOUS FAILURE DIAGNOSIS OF DISCRETE-EVENT SYSTEMS

Felipe Gomes de Oliveira Cabral

October/2017

Advisor: Marcos Vicente de Brito Moreira

Department: Electrical Engineering

In general, systems are formed by the composition of several modules, local components or subsystems, and may exhibit a large number of states. The growth of the global system model with the number of system components leads to high computational costs for failure diagnosis techniques based on the global model. In this work, a new approach for the failure diagnosis of discrete event systems is introduced. The method is based on the computation of a Petri net diagnoser, called synchronized Petri net diagnoser (SPND), that is constructed from the nonfailure behavior of the modules of the system. We also introduce the definition of synchronous diagnosability of the language of a system with respect to the languages of its modules, and present an algorithm to verify this property. We also propose a decentralized synchronized Petri net diagnosis scheme for discrete-event systems modeled as automata. In order to do so, we define the notion of synchronous codiagnosability and propose an algorithm to verify this property. The synchronized diagnosis can be refined using the global nonfailure model of the system, leading to the notion of conditional synchronous diagnosability. An algorithm for the verification of conditional synchronous diagnosability based on the verification of synchronous diagnosability is proposed. We also discuss the relation among conditional synchronous diagnosability, synchronous codiagnosability, synchronous diagnosability and diagnosability of discrete-event systems. Algorithms for the computation of the maximum delay bound for all diagnosis schemes presented in this work are proposed. An example and a practical implementation of the diagnosis methods are presented and used throughout this work in order to illustrate and validate the methods.

Contents

List of Figures x

List of Symbols xiii

1 Introduction 1

2 Fundamentals of Discrete-Event Systems 8

2.1 Languages . . . 8

2.1.1 Language operations . . . 9

2.2 Automata . . . 11

2.2.1 Operations on automata . . . 13

2.2.2 Automata with partially observed events . . . 15

2.3 Petri nets . . . 18

2.3.1 Petri net structure . . . 18

2.3.2 Petri net marking . . . 19

2.3.3 Petri net dynamics . . . 20

2.3.4 Labeled Petri net . . . 21

2.3.5 State machine Petri net . . . 21

2.3.6 Binary Petri net . . . 23

2.3.7 Extended Petri net . . . 23

2.4 Diagnosability of DESs . . . 24

2.4.1 Centralized diagnosability of DESs . . . 24

2.4.2 Codiagnosability of DESs . . . 33

2.4.3 Modular diagnosability of DESs . . . 35

2.5 Final remarks . . . 39

3 Synchronous centralized diagnosability of DESs 41

3.1 Synchronous diagnosability . . . 42

3.2 Synchronous diagnosability verifier . . . 46

3.3 Delay bound for synchronous diagnosis . . . 52

3.3.1 Complexity analysis . . . 59

3.4 Synchronized Petri net diagnoser . . . 60

3.5 Synchronized Petri net diagnoser for an automated system 64
3.5.1 Case study system 64
3.5.2 Modeling the controlled plant 65
3.5.3 Synchronized Petri net diagnoser 68
3.6 Final remarks 70

4 Synchronous codiagnosability of DESs **75**
4.1 Synchronous codiagnosability 75
4.2 Synchronous decentralized failure diagnosis 82
4.3 Comparison between modular diagnosability and synchronous codiagnosability 86
4.4 Final remarks 91

5 Conditional synchronous diagnosability of DESs **93**
5.1 Conditional synchronous Petri net diagnoser 98
5.2 Conditional synchronous diagnosability 102
5.3 Conditional synchronized Petri net diagnoser for an automated system 108
5.4 Final remarks 111

6 Conclusion and future research topics **115**

Bibliography **119**

List of Figures

1.1 Comparison between the main diagnosis architectures proposed in the literature: the monolithic scheme (a); the decentralized scheme (b); the distributed scheme (c); and the modular scheme (d). 5

1.2 Main diagnosis schemes presented in the literature and the synchronous diagnosis scheme proposed in this work. 6

2.1 State transition diagram of automaton G of Example 2.1. 12

2.2 Automata G_1 and G_2 of Example 2.2. 16

2.3 Automata G_{prod} and G_{par} of Example 2.2. 16

2.4 State transition diagram of automaton G of Example 2.3 (a), and observer automaton of G, $Obs(G, \Sigma_o)$, that provides the state estimates of G after the observation of a trace generated by the system (b). . . 18

2.5 Structure of the Petri net of Example 2.4. 19

2.6 Two examples of Petri nets with different initial markings. 20

2.7 Petri net of Example 2.6 with transition t_1 enabled (a), and after the firing of transition t_1 with the new reached marking (b). 21

2.8 Labeled Petri net of Example 2.7. 22

2.9 Automaton G (a), and state machine Petri net $\mathcal{N}$ (b) of Example 2.8. 22

2.10 Petri net $\mathcal{N}$ of Example 2.9 with transition t_1 enabled (a), and after the firing of transition t_1 with the new reached marking (b). Petri net $\mathcal{N}$ with a different marking where transition t_1 is not enabled (c). 24

2.11 Automaton G (a), automaton G_l (b), and diagnoser automaton G_d (c) of Example 2.10. 27

2.12 Automaton A_l of Example 2.10. 27

2.13 Diagnoser automaton G'_d considering Σ'_o as the set of observable events of Example 2.10. 28

2.14 Nonfailure behavior automaton G_N of the system G of Example 2.11. 31

2.15 State machine Petri net $\mathcal{N}$ of Example 2.11. 31

2.16 State observer Petri net $\mathcal{N}_{SO}$ of Example 2.11. 32

2.17 Petri net diagnoser $\mathcal{N}_D$ of Example 2.11. 32

2.18 Automaton G of Example 2.12. 35

2.19 Automaton G_N (a) and automaton G_F (b) of Example 2.12. 36
2.20 Automaton G_V of Example 2.12. 36
2.21 Automata G_1, G_2 and G_3 of Example 2.13. 39
2.22 Automaton G of Example 2.13. 39
2.23 Diagnoser automaton G_{d_1} of Example 2.13. 39
2.24 Diagnoser automaton G_d of Example 2.13. 40

3.1 Comparison between the monolithic diagnosis architecture (a); and the synchronous diagnosis architecture (b). 42
3.2 Automata G_1 and G_2 of Example 3.1. 47
3.3 Automaton G of Example 3.1. 47
3.4 Automaton G_N of Example 3.1. 48
3.5 Automata G_{N_1} and G_{N_2} of Example 3.1. 49
3.6 Automaton G_F of Example 3.2. 53
3.7 Automata $G^R_{N_1}$ and $G^R_{N_2}$ of Example 3.2. 53
3.8 Automaton G^R_N of Example 3.2. 54
3.9 Automaton G^{SD}_V of Example 3.2. 55
3.10 Graph $\overline{G}^{SD}_V = G_{dag}$ of Example 3.3. 58
3.11 Graph G_{dag} of Example 3.3 topologically sorted. 59
3.12 Graph G_{dag} of Example 3.3 topologically sorted with the value of the weighting functions $\rho(v_i, v_j)$ (above the edges) and $l(v_j)$ (below the vertices). 59
3.13 Petri nets $\mathcal{N}_{D_1}$ and $\mathcal{N}_{D_2}$ of Example 3.4. 63
3.14 Synchronized Petri net diagnoser $\mathcal{N}_D$ of Example 3.4. 64
3.15 Schematics of the conveyor belt and the handling unit considered in the case study. 65
3.16 Automaton G_{cb} that models the conveyor belt (a); and automaton G_{hu} that models the handling unit (b). 66
3.17 Automaton G_F obtained from G_p. 71
3.18 Automaton $G_{N_{cb}}$ (a); and automaton $G_{N_{hu}}$ (b). 72
3.19 Automaton $G^R_{N_{cb}}$ (a); and automaton $G^R_{N_{hu}}$ (b). 72
3.20 Petri net diagnoser $\mathcal{N}_{D_p}$. 73

4.1 Comparison between the decentralized diagnosis architecture (a); and the synchronous decentralized diagnosis architecture (b). 76
4.2 Automaton G of Example 4.1. 78
4.3 Automata G_1 and G_2 of Example 4.1. 79
4.4 Automata G_{N_1} and G_{N_2} of Example 4.1. 79
4.5 Relation between the notions of diagnosability, synchronous diagnosability and synchronous codiagnosability. 79

4.6 Automata G_1 and G_2 of Example 4.2. 81
4.7 Automata $\hat{G}^R_{N_1}$ and $\hat{G}^R_{N_2}$ of Example 4.2. 82
4.8 Automaton $\hat{G}^R_N$ of Example 4.2. 83
4.9 Part of automaton G^{SC}_V formed by the states labeled with F and their related transitions of Example 4.2. 84
4.10 Automata G_{N_1} and G_{N_2} of Example 4.4. 85
4.11 Local Petri net diagnosers $\mathcal{N}_1$ and $\mathcal{N}_2$ of Example 4.4. 86
4.12 Automata G_1, G_2 and G_3 of Example 4.5. 90
4.13 Automaton G of Example 4.5. 90
4.14 Automaton G_F of Example 4.5. 90
4.15 Automaton G^R_N of Example 4.5. 91
4.16 Automaton G^M_V of Example 4.5. 91
4.17 Local Petri net diagnoser $\mathcal{N}_1$ of Example 4.5. 91

5.1 Automata G_1 and G_2 of Example 5.1. 94
5.2 Automaton G of Example 5.1. 95
5.3 Automaton G_N of Example 5.1. 95
5.4 Automaton G^R_N of Example 5.1. 96
5.5 Synchronized Petri net diagnoser $\mathcal{N}_D$ of Example 5.1. 97
5.6 Automata G_{N_1} and G_{N_2} of Example 5.1. 97
5.7 State observer Petri nets $\mathcal{N}_{SO_1}$ and $\mathcal{N}_{SO_2}$ of Example 5.3. 99
5.8 Conditional Petri net state observer $\mathcal{N}^c_{SO_1}$ of Example 5.3. 100
5.9 Conditional Petri net state observer $\mathcal{N}^c_{SO_2}$ of Example 5.3. 101
5.10 Automaton $G^R_{N,c}$ of Example 5.4. 104
5.11 Automaton G_F of Example 5.5. 105
5.12 Verifier automaton $G^{SD}_{V,c}$ of Example 5.5. 106
5.13 Automaton G_1 (a); and automaton G_2 (b) of Example 5.6. 109
5.14 Automaton G_F of Example 5.6. 109
5.15 Automaton G^R_N of Example 5.6. 109
5.16 Automaton $G^R_{N,c}$ of Example 5.6. 110
5.17 Conditional synchronized Petri net diagnoser $\mathcal{N}_{D_p,c}$. 112
5.18 Petri net diagnoser $\mathcal{N}_{D_p}$. 113

List of Symbols

$Ac(G)$	Accessible part of G, p. 12
$CoAc(G)$	Coaccessible part of G, p. 12
G	Automaton, p. 10
$G_{V,c}^{SD}$	Conditional synchronous verifier automaton, p. 103
G_N	Automaton that models the nonfailure behavior of the system G, p. 24
G_N^R	Automaton resultant from the parallel composition of automata $G_{N_k}^R$, p. 47
G_V	Codiagnosability verifier automaton, p. 32
G_V^{SD}	Synchronous diagnosis verifier automaton, p. 47
G_d	Diagnoser automaton, p. 24
G_k	Automaton model of the k-th component of G, p. 39
G_p	Global plant model of the cube assembly mechatronic system, p. 63
G_{N_k}	Nonfailure behavior model of the k-th component of G, p. 39
$G_{N_k}^R$	Automaton G_{N_k} with its unobservable events renamed, p. 47
G_V^{SC}	Synchronous codiadnosability verifier, p. 77
G_{cb}	Conveyor belt model, p. 63
G_{dag}	Directed acyclic graph, p. 55
G_{hu}	Handling unit model, p. 63
$I(p_i)$	Set of input transitions of place p_i, p. 17

$I(t_j)$	Set of input places of transition t_j, p. 17
L	Generated language of automaton G, p. 11
L_F	Failure language, p. 24
L_N	Generated language of automaton G_N, p. 24
L_a	Augmented language, p. 43
L_{N_a}	Augmented nonfailure language, p. 43
L_{N_k}	Nonfailure language of automaton G_k, p. 43
$Obs(G, \Sigma_o)$	Observer automaton of G in Σ_o, p. 16
$Out(p_i)$	Set of output transitions of place p_i, p. 17
$Out(t_j)$	Set of output places of transition t_j, p. 17
P	Set of places of a Petri net, p. 17
P_o	Projection operation defined as $P_o : \Sigma^\star \to \Sigma_o^\star$, p. 15
P_s^l	Projection operation defined as $P_s^l : \Sigma_l^\star \to \Sigma_s^\star$, p. 9
P_{o_i}	Projection defined as $P_{o_i} : \Sigma^\star \to \Sigma_{o_i}^\star$, p. 31
$Post$	Function of arcs that connect transitions to places in a Petri net, p. 17
Pre	Function of arcs that connect places to transitions in a Petri net, p. 17
Q	Set of states, p. 10
Q_m	Set of marked states, p. 10
R_k	Renaming function defined as $R_k : \Sigma_{N_k} \to \Sigma_{N_k}^R$, p. 47
T	Set of transitions of a Petri net, p. 17
$UR(q)$	Unobservable reach of state q, p. 15
Γ_G	Feasible event function of automaton G, p. 10
Σ	Set of events, p. 7
Σ_f	Set of failure events, p. 24

Σ_i	Event set of the i-th site, p. 31
Σ_k	Set of events of automaton G_k, p. 39
Σ_o	Observable event set, p. 15
$\Sigma_{i,uo}$	Set of unobservable events of the i-th local diagnoser, p. 31
$\Sigma_{k,o}$	Observable event set of G_k in the synchronous centralized diagnosis scheme, p. 39
$\Sigma_{k,uo}$	Unobservable event set of G_k in the synchronous centralized diagnosis scheme, p. 39
Σ_{o_i}	Set of observable events of the i-th local diagnoser, p. 31
Σ_{uo}	Unobservable event set, p. 15
ℓ	Total number of sites of a system G, p. 31
$\hat{L}_{N_a}$	Augmented nonfailure language for synchronous decentralized diagnosis, p. 75
$\hat{P}_{k,o}$	Projection operation defined as $\hat{P}_{k,o} : \Sigma^\star \rightarrow \hat{\Sigma}^\star_{k,o}$, p. 74
$\hat{\Sigma}_{k,o}$	Observable event set of G_k in the synchronous decentralized diagnosis scheme, p. 73
$\hat{\Sigma}_{k,uo}$	Unobservable event set of G_k in the synchronous decentralized diagnosis scheme, p. 73
$\lvert . \rvert$	Set cardinality, p. 17
$\mathbb{N}$	Natural numbers set, including the number zero, p. 17
$\mathcal{L}(G)$	Generated language of automaton G, p. 11
$\mathcal{L}_m(G)$	Marked language of automaton G, p. 11
$\mathcal{N}$	Petri net, p. 18
$\mathcal{N}^c_{SO_k}$	Conditional Petri net observer, p. 96
$\mathcal{N}_D$	Synchronized Petri net diagnoser, p. 58
$\mathcal{N}_k$	Local Petri net diagnoser, p. 73
$\mathcal{N}_k$	Local Petri net diagnoser for synchronous decentralized diagnosis, p. 82

$\mathcal{N}_{D,c}$	Conditional Petri net diagnoser, p. 98
$\mathcal{N}_{SO_k}$	Petri net state observer of G_{N_k}, p. 40
μ	Cardinality of set T, p. 17
ν	Cardinality of set P, p. 17
$\overline{L}$	Prefix-closure operation on language L, p. 8
$\parallel s \parallel$	Length of trace s, p. 7
$\parallel$	Parallel composition, p. 14
$\setminus$	Set difference, p. 9
$\star$	Kleene-closure operation, p. 8
$\times$	Product composition, p. 13
ε	Empty trace, p. 7
$\underline{x}$	Marking vector, p. 18
f	Transition function, p. 10
l	Transition labeling function, p. 20
q_0	Initial state, p. 10
x	Marking function defined as $x : P \to \mathbb{N}$, p. 18
x_0	Initial marking function, p. 18
z^*	Delay bound for synchronous diagnosis, p. 54

Chapter 1

Introduction

Systems that have a discrete state space and whose evolution is driven by the occurrence of events, and are not time-driven, are called Discrete-Event Systems (DESs) [1, 2]. Events are characterized by an instantaneous occurrence, such as the beginning or the finishing of a task, a sensor state change, or the pressing of a button by an operator. Several systems can be modeled as DESs, such as manufacturing systems, robotic systems, traffic supervision, data management, logistic, and energy systems.

A DES cannot be modeled by differential or difference equations due to its discrete nature, and the fact that its evolution is given by the occurrence of events. Thus, a different mathematical formalism is necessary to describe DESs, and the most common used in the literature are automata and Petri nets [1–5]. Automata are directed graphs, where the vertices represent the states of the system, and the arcs represent transitions labeled with events in order to model their occurrence. Petri nets are bipartite graphs, or bigraphs, in the sense that it has two types of nodes (places and transitions), where nodes of the same type cannot be connected. Tokens are assigned to the places of the Petri net, such that the number of tokens of each place forms the marking of the Petri net, which also represents the system state modeled by the net. Notice that, differently from automata, the state of the system is represented in a distributed way and, because of this property, Petri nets are usually used to represent systems with a high degree of concurrency and a large number of states.

Automatic systems are becoming more and more independent from human interaction and thus, more complex. Such complexity can be seen in the increase of systems that are composed of several subsystems that interact in order to complete tasks. When modeled by an automaton, the global plant model of a DES is obtained by composing the automaton models of its subsystems. The state space of the resulting automaton can grow exponentially with the number of subsystems, which can prevent the application of feedback control techniques, known as supervisory

control, with a view to modifying the behavior of the system in order to achieve a set of specifications [1, 6, 7]. In order to avoid the use of the global system model for supervisory control, local modular control strategies have been proposed in the literature [8–13]. In this work, we take advantage of the modularity of systems in order to investigate a different problem: the failure diagnosis of DESs.

Automatic systems are subject to failures that can alter their expected behavior and decrease their reliability and performance. Therefore, the study of failure diagnosis techniques of DESs are fundamental in order to identify the occurrence of a failure. Usually, a failure is modeled as an unobservable event, *i.e.*, its occurrence cannot be detected by a sensor, and, in order to identify if a failure event has occurred, it is necessary to build a DES model of the nonfailure and post-failure behaviors of the system. Then, the failure occurrence can be diagnosed by following the observed traces generated by the system. Several works in the literature address the problem of failure diagnosis of discrete-event systems (DESs) modeled by automata [14–28], timed automata [29–31], and Petri nets [32–39]. In ZAYTOON and LAFORTUNE [40], an overview of the diagnosis methods for DESs presented in the literature is carried out.

In the seminal work of SAMPATH *et al.* [14, 15], a model based failure diagnosis scheme is proposed for DESs, and an automaton diagnoser, whose states are state estimates of the system reached after the observation of a trace, is presented. Although the diagnoser presented in SAMPATH *et al.* [14, 15] can be straightforwardly implemented for failure diagnosis, its construction is, in general, avoided since, in the worst-case, the state-space of the diagnoser grows exponentially with the cardinality of the state-space of the plant model [14, 15, 19, 41]. Recently, in CLAVIJO and BASILIO [42], an empirical study on the average state-space size of the diagnoser proposed in SAMPATH *et al.* [14] is carried out. In CLAVIJO and BASILIO [42] it is shown that, on the average, the state-space cardinality of the diagnoser proposed in SAMPATH *et al.* [14] can grow polynomially in the number of states of the system.

In SAMPATH *et al.* [14], it is stated that diagnosis can be carried out storing only the current state of the diagnoser, without the need for storing the complete state space of the diagnoser, and, after the observation of an event, update the state estimate. However, a method for this implementation is not presented in SAMPATH *et al.* [14]. In QIU and KUMAR [19], a method for diagnosis that avoids the construction of the diagnoser automaton, is presented. In order to do so, a nondeterministic automaton is computed, and only the current state of the diagnoser and the nondeterministic automaton need to be stored. After the occurrence of an observable event, the next state of the diagnoser can be computed online in polynomial time. However, the details of the practical implementation on a computer are

not presented in QIU and KUMAR [19].

The diagnosis methods presented in SAMPATH *et al.* [14], CARVALHO *et al.* [23, 25], CABRAL *et al.* [26], SANTORO *et al.* [28], CABASINO *et al.* [38] consider that all system information regarding failure diagnosis, *e.g.*, sensor signals, is available in a centralized way. However, there is a large number of systems where the diagnosis information is only available locally [17], which makes the decentralized [17, 19, 20, 43] and distributed [44, 45] architectures more appropriated for such systems. In DEBOUK *et al.* [17], several protocols for decentralized diagnosis are proposed. The notion of diagnosability introduced in SAMPATH *et al.* [14] is extended to decentralized architectures, consisting of local diagnosers that communicate with a coordinator, in order to detect failure event occurrences. Several protocols for decentralized diagnosis, that determine the diagnostic information generated at each local site, the communication rules used by the local sites, and the decision rule for failure diagnosis applied by the coordinator are presented in DEBOUK *et al.* [17].

In Protocol 3 of DEBOUK *et al.* [17], the local diagnosers do not communicate among each other, and the inference on the occurrence of the failure event is carried out based solely on their own observations. When at least one of the local diagnosers identifies the failure event occurrence, the diagnostic is sent to a coordinator that informs it to the system operator. This notion of decentralized diagnosability has been referred to as disjunctive-codiagnosability [20]. The diagnosability notion presented in SAMPATH *et al.* [14] is a particular case of the disjunctive-codiagnosability case when only one local diagnoser is considered [17]. A different notion of decentralized diagnosability has been defined in WANG *et al.* [20] and YAMAMOTO and TAKAI [46], and it is called conjunctive-codiagnosability. In this architecture, any non-failure trace can be distinguished from the failure traces, after a bounded number of event occurrences, by at least one local diagnoser. The conjunctive-codiagnosability and disjunctive-codiagnosability are incomparable [20], which means that a system can be conjunctive-codiagnosable and not disjunctive-codiagnosable, and vice-versa. In this work, we are interested only in the disjunctive decentralized diagnosis, which, from this point, is referred to as codiagnosability.

A vast range of diagnosis methods can be found in the literature for systems modeled as Petri nets. The simplest way to perform diagnosis in systems modeled as Petri nets is to build the reachability graph of the Petri net that models the system, and, after that, obtain its diagnoser. In practice, this approach implies in replacing the Petri net model with an automaton model of the system, and the benefit to represent the state of the system in a distributed way in the net is lost. In fact, the graph of an automaton model can be much larger than the graph of a Petri net model for the same system. In order to overcome this problem, several diagnosis

methods for systems modeled as Petri nets have been proposed in the literature [35, 47–51].

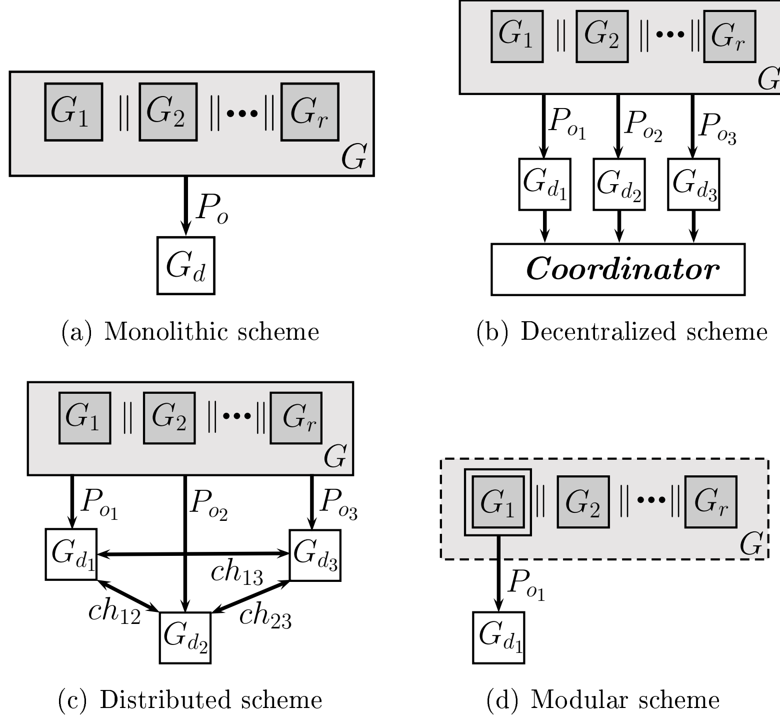
Recently, in CABRAL *et al.* [26], a new approach for failure diagnosis of systems modeled as finite state automata is proposed. The diagnosis method is based on the construction of a Petri net diagnoser (PND), which can be obtained in polynomial time, and provides the current state estimate of the non-failure part of the system model after the observation of a trace. If an observed trace is executed by the system, and it is not in the nonfailure behavior model, then a failure is detected. Alternative diagnosis approaches that only consider the nonfailure behavior of the system can also be found in the literature [52–55].

In all methods presented in [14–16, 19, 20, 26] the diagnosers are computed based on the global plant model, which can grow exponentially with the number of system components. In order to circumvent this problem, failure diagnosis schemes have been proposed for systems with a modular structure [56–63]. The main idea in these works is to exploit the modular structure of the system with a view to reducing the cost associated with the computation of the global system model for diagnosis.

In DEBOUK *et al.* [56] and CONTANT *et al.* [59], different notions of modular diagnosability are proposed. In these works, it is assumed that the failure event is modeled in a single component of the system, and the goal is to identify the occurrence of the failure event by using only this component model instead of the global system model. In CONTANT *et al.* [59], it is assumed that the module where the failure event is modeled has a persistent excitation, which allows that languages that are not diagnosable using the classical definition of diagnosability presented in SAMPATH *et al.* [14], be modularly diagnosable. Moreover, it is also assumed in CONTANT *et al.* [59] that the system has no cycles of unobservable events, and that the common events of the modules are observable, which implies that the failure event belongs only to the event set of the component used to construct the diagnoser.

In PENCOLÉ and CORDIER [58], a different modular diagnosis approach is proposed. In this work, a local diagnoser is constructed for each component of the system and the local diagnoses are merged in order to obtain the global diagnosis decision. The main drawback of this method is that, in the worst-case, the paths of all modules of the system must be synchronized, which leads to an exponential growth with the number of system components.

In GARCÍA *et al.* [64], a different approach for modular diagnosis of DESs is proposed. Differently from [56, 59–61], the method proposed in GARCÍA *et al.* [64] consists in splitting the system into subsystems, constructing a minimum local controller for each subsystem, and then computing a diagnoser for each subsystem composed with its minimum local controller. The failure event is detected when a local diagnoser identifies its occurrence. In SCHMIDT [62], an incremental



(a) Monolithic scheme (b) Decentralized scheme

(c) Distributed scheme (d) Modular scheme

Figure 1.1: Comparison between the main diagnosis architectures proposed in the literature: the monolithic scheme (a); the decentralized scheme (b); the distributed scheme (c); and the modular scheme (d).

abstraction-based approach for the verification of modular language diagnosability of DESs is proposed, and the differences between the diagnosis methods presented in [56, 58–60] are reviewed.

In Figure 1.1 we show the schematics of the main diagnosis architectures proposed in the literature: (*i*) the monolithic scheme; (*ii*) the decentralized scheme; (*iii*) the distributed scheme; and (*iv*) the modular scheme. Notice that in the modular architecture we consider that the failure component is G_1 and, thus, only the local diagnoser G_{d_1} is implemented.

In this work, we propose a new scheme for centralized failure diagnosis and decentralized failure diagnosis of DESs. We first propose a method for centralized diagnosis that avoids the computation of the global system model. The method is based on the computation of a Petri net diagnoser, called synchronized Petri net diagnoser (SPND), which is constructed from the nonfailure behavior of the system modules. The SPND carries out the online synchronization of the system modules in order to provide a set of states that contains the state estimate of the nonfailure behavior of the global system. If the observation of a trace is not recognized in the SPND, the occurrence of the failure event is detected. In this context, we introduce the definition of synchronous diagnosability of the language of a DES with respect to the languages of its modules, and present an algorithm to verify this property.

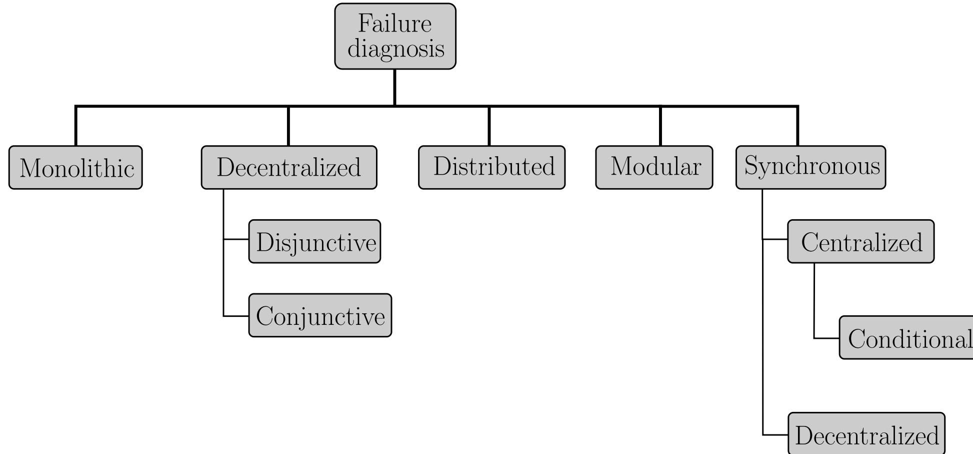


Figure 1.2: Main diagnosis schemes presented in the literature and the synchronous diagnosis scheme proposed in this work.

We also propose a decentralized architecture for diagnosis of DESs. This is done by extending the notion of synchronous diagnosis to the decentralized case using a scheme similar to the one presented in Protocol 3 of DEBOUK *et al.* [17], where all information regarding the observation of events are available locally. In order to do so, we assume that each component of the system has its own set of observable events, and a local diagnoser is implemented for each module. We introduce the definition of synchronous codiagnosability, and discuss a verification method and the implementation of this scheme. Moreover, we present a method for the computation of the maximum delay bound for synchronous decentralized diagnosis that can also be used for the synchronous diagnosis method.

Since the state estimate of the SPND contains the state estimate of the nonfailure behavior of the global system, the synchronous diagnosis is equivalent to the diagnosis of a system with an augmented nonfailure language. The synchronous diagnosis of DESs can be refined with the view to reducing the nonfailure language for synchronous diagnosis. This refinement consists in the addition of boolean conditions to the transitions of the SPND, which leads to the Conditional Synchronized Petri Net Diagnoser (CSPND). Since the nonfailure language for synchronous diagnosis is reduced, the notion of conditional synchronous diagnosability is introduced and an algorithm to verify this property is presented. We show that systems that are not synchronously diagnosable can be conditionally synchronously diagnosable and the delay bound for synchronous diagnosis can be decreased using the conditional synchronous diagnosis scheme. In Figure 1.2, we show the main diagnosis architectures presented in the literature. Notice that the synchronous diagnosis scheme proposed in this work is a new architecture and can be implemented in a centralized or decentralized way.

We validate all diagnosis methods presented in this work by applying then to a didactic manufacturing system [65, 66]. The manufacturing system consists of a cube assembly mechatronic plant located at the Control and Automation Laboratory (LCA) of the Federal University of Rio de Janeiro (UFRJ). We show how the controlled behavior of the plant is modeled for the application of the synchronous diagnosis methods. The Petri net diagnosers are presented, and the failure diagnosis process is illustrated. The delay bound for each synchronous diagnosis method is also computed.

This work is organized as follows. In Chapter 2, we present preliminary concepts about DESs modeled as automata and Petri nets. We also present a theoretical background of failure diagnosis of systems modeled as automata. In Chapter 3, we present the synchronous centralized diagnosis scheme, and introduce the definition of synchronous centralized diagnosability of DESs. A verification method of the synchronous diagnosability of DESs and an algorithm to compute the maximum delay bound for synchronous diagnosis are introduced. In Chapter 4, we present the synchronous decentralized diagnosis scheme, and the definition of synchronous decentralized diagnosability of DESs. We also show a comparison between the notions of synchronous codiagnosability, synchronous diagnosability, the classical diagnosability, and modular diagnosability of DESs. The conditional synchronous diagnosis architecture is presented in Chapter 5, where the conditional synchronized Petri net diagnoser is proposed. A comparison between all notions of synchronous diagnosability is also carried out in Chapter 5. Finally, in Chapter 6, we present the conclusions of this work, together with future research topics related to this thesis.

Chapter 2

Fundamentals of Discrete-Event Systems

A Discrete Event System (DES) is a system whose state-space is a discrete set and whose evolution is driven by the occurrence of events. Thus, DESs cannot be described by differential or difference equations and, therefore, it is necessary to present mathematical formalisms that are capable of correctly representing the evolution of a DES. Although a DES can be described only by its language, this representation is not practical. In this work, we consider two types of modeling formalisms largely used to describe DESs: automata and Petri nets [1–5].

In order to present the concepts of automata and Petri nets, we first present the concept of languages, and some notations and definitions.

2.1 Languages

In this work, we use the symbol Σ to represent the set of events of a given DES. The symbol σ is used to represent a generic event. A sequence of events forms a trace and a trace consisting of no events is called the empty trace and it is represented by ε. If s is a trace, its length is denoted by $\|s\|$. The length of the empty trace ε is considered to be zero. The formal definition of a language is presented as follows [1].

Definition 2.1 (Language) *A language L defined over a set of events Σ is a set of finite length traces formed with the events of Σ.*

For example, the language $L = \{\varepsilon, e, ed, dee, eed\}$ is composed of five traces, including the empty trace ε, formed with events of $\Sigma = \{d, e\}$. It is important to remark that languages are sets and, therefore, all operations of sets can also be applied to languages. In the following, we present other operations that can be executed using events and traces with the aim to create and modify languages.

2.1.1 Language operations

The main operation related to the construction of traces from a set of events Σ, and therefore languages, is the concatenation. Consider, for example, the trace abc, formed with the events of $\Sigma = \{a, b, c\}$. The trace abc can be formed by the concatenation of trace ab with event c. Notice that ab is, itself, a concatenation of the events a and b. The empty trace ε is the identity element of the concatenation operation, *i.e.*, $\sigma\varepsilon = \varepsilon\sigma = \sigma$.

A language defined over Σ is a subset of the set formed by all finite length traces of events built with the elements of Σ, including the empty trace ε. This set is denoted by $\Sigma^\star$, where the operation $\star$ is called Kleene-closure. In particular, the sets $\emptyset$, Σ and $\Sigma^\star$ are also languages.

The concatenation and Kleene-closure operations can also be defined for languages, as it is presented in the sequel.

Definition 2.2 (Concatenation) *Let $L_1, L_2 \subseteq \Sigma^\star$, then the concatenation L_1L_2 is defined as:*

$$L_1L_2 = \{s = s_1s_2 : (s_1 \in L_1) \textit{ and } (s_2 \in L_2)\}.$$

A trace s is in L_1L_2 if it is formed by the concatenation of a trace $s_1 \in L_1$ and $s_2 \in L_2$.

Definition 2.3 (Kleene-closure) *Let $L \subseteq \Sigma^\star$, then*

$$L^\star = \{\varepsilon\} \cup L \cup LL \cup \ldots$$

An element of $L^\star$ is formed by the concatenation of elements of L. By definition, the empty trace ε is also an element of $L^\star$, representing the concatenation of "zero" elements. Moreover, the Kleene-closure operation is idempotent, *i.e.*, $(L^\star)^\star = L^\star$.

Another important operation that can be applied to languages is the Prefix-closure. Before we present this operation, it is necessary to define prefix, subtrace and suffix of a trace s. Let $s = tuv$, where $t, u, v \in \Sigma^\star$, then t is the prefix of s, u is the subtrace of s, and v is the suffix of s. Since $t, u, v \in \Sigma^\star$, then the traces ε and s are also prefixes, subtracess and suffixes of s. The prefix-closure of a language L is formally defined as follows.

Definition 2.4 (Prefix-closure) *Let $L \subseteq \Sigma^\star$, then*

$$\overline{L} = \{s \in \Sigma^\star : (\exists t \in \Sigma^\star)[st \in L]\}.$$

The prefix-closure of a language L, denoted as $\overline{L}$, is the set of all prefixes of all traces of L. Notice that, by definition, $L \subseteq \overline{L}$. A language L is said to be prefix-closed if $L = \overline{L}$, *i.e.*, if all prefixes of all traces of L are also elements of L.

Remark 2.1 *It is important to remark that, for a language* $L = \emptyset$, $\overline{L} = \emptyset$. *However, if* $L \neq \emptyset$, *then* $\varepsilon \in \overline{L}$. *Moreover,* $\emptyset^\star = \{\varepsilon\}$ *and* $\{\varepsilon\}^\star = \{\varepsilon\}$, *and the concatenation operation between a language and the empty set is equal to the empty set, i.e.,* $\emptyset L = L\emptyset = \emptyset$.

Another important operation that can be applied to traces, or languages, is the projection operation, defined as follows [1].

Definition 2.5 (Projection) *The natural projection* $P_s^l : \Sigma_l^\star \to \Sigma_s^\star$, *where* $\Sigma_s \subset \Sigma_l$, *is defined recursively as follows:*

$$P_s^l(\varepsilon) = \varepsilon,$$

$$P_s^l(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_s, \\ \varepsilon, & \text{if } \sigma \in \Sigma_l \setminus \Sigma_s, \end{cases}$$

$$P_s^l(s\sigma) = P_s^l(s)P_s^l(\sigma), \text{ for all } s \in \Sigma_l^\star, \sigma \in \Sigma_l,$$

where $\setminus$ *denotes set difference.*

According to Definition 2.5, the projection operation erases all events $\sigma \in \Sigma_l \setminus \Sigma_s$ from the traces $s \in \Sigma_l^\star$. The inverse projection operation is defined as follows.

Definition 2.6 (Inverse projection) *The inverse projection* $P_s^{l^{-1}} : \Sigma_s^\star \to 2^{\Sigma_l^\star}$ *is defined as:*

$$P_s^{l^{-1}}(t) = \{s \in \Sigma_l^\star : P_s^l(s) = t\}.$$

For a given trace t, formed with events from Σ_s, $P_s^{l^{-1}}(t)$ produces a set formed with all traces that can be constructed with the events of Σ_l whose projection P_s^l is equal to t.

The operations P_s^l and $P_s^{l^{-1}}$ can be extended to languages. In order to do so, it is necessary to apply these operations to all traces that belong to the language. The main application of the projection is to represent the observed language of a system obtained from an observer that has access only to the events registered by sensors or command events sent by a controller.

It is important to notice the following property obtained from the definition of projection, and set theory:

$$P_s(L_a \cap L_b) \subseteq P_s(L_a) \cap P_s(L_b), \tag{2.1}$$

where L_a and L_b are two languages defined over a set of events Σ, and $P_s : \Sigma^\star \to \Sigma_s^\star$, where $\Sigma_s \subset \Sigma$.

The language of a DES is a set that contains the information regarding all admissible traces that a system is capable of generating. Using languages to describe DESs can be a difficult task, since, depending on the system, it is not easy to represent all its behavior by describing in a set all possibilities of traces generated by the system. Therefore, it is necessary to define a structure that is capable of representing the language of a system and that can be manipulated by using well defined operations, allowing the construction and analysis of systems that generate complex arbitrarily languages. In the next section, we define one of the formalisms used in this work to represent languages.

2.2 Automata

An automaton is a device that is capable of representing a language according to well-defined rules [1, 2]. In the following, we formally define an automaton.

Definition 2.7 (Automaton) *An automaton, denoted by G, is a five-tuple*

$$G = (Q, \Sigma, f, q_0, Q_m),$$

where Q is the set of states, Σ is the set of events, $f : Q \times \Sigma \to Q$ is the transition function, q_0 is the initial state, and Q_m is the set of marked states.

The transition function f describes all transitions of the automaton, such that $f(q_1, \sigma) = q_2$ means that there is a transition (q_1, σ, q_2), *i.e.*, there exists a transition from state q_1 to state q_2 labeled with event σ. For the sake of simplicity, the set of marked states Q_m will be omitted from the automata defined in this work, unless stated otherwise. In other words, an automaton may be represented by a four-tuple $G = (Q, \Sigma, f, q_0)$, which implies that the set of marked states is $Q_m = \emptyset$.

We also define $\Gamma_G : Q \to 2^{\Sigma}$ as the feasible event function of a state of G. The feasible event function $\Gamma_G(q)$ is the set of all events σ for which $f(q, \sigma)$ is defined. Notice that the feasible event function Γ_G can be completely described from the transition function f.

An automaton can be represented graphically by an oriented graph called state transition diagram. The states and transitions of the automaton are represented by circles, forming the vertices of the graph, and oriented arcs connecting the states, respectively. The arcs are labeled with the events of Σ that cause the transition of states. In order to represent the initial state of the automaton, we add an arc that does not have an origin state attached to it.

In the following, we present an example of an automaton and its state transition diagram.

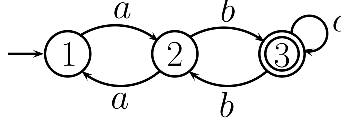


Figure 2.1: State transition diagram of automaton G of Example 2.1.

Example 2.1 *Let G be an automaton whose state transition diagram is depicted in Figure 2.1. The state and event sets of G are given by $Q = \{1, 2, 3\}$ and $\Sigma = \{a, b, c\}$, respectively. The feasible event function is defined as: $\Gamma_G(1) = \{a\}$, $\Gamma_G(2) = \{a, b\}$, and $\Gamma_G(3) = \{b, c\}$. The transition function of G is defined as: $f(1, a) = 2$, $f(2, b) = 3$, $f(3, c) = 3$, $f(3, b) = 2$, and $f(2, a) = 1$. The initial state q_0 of G is 1, and the set of marked states is $Q_m = \{3\}$.*

In this work, we define a path of an automaton G as a sequence $(q_1, \sigma_1, q_2, \ldots, q_{n-1}, \sigma_{n-1}, q_n)$, where $\sigma_i \in \Sigma$, $q_{i+1} = f(q_i, \sigma_i)$, $i = 1, 2, \ldots, n-1$. A path $(q_1, \sigma_1, q_2, \ldots, q_{n-1}, \sigma_{n-1}, q_n)$ is a cyclic path, or simply a cycle, if $q_1 = q_n$.

In the following, we define the generated and marked languages of an automaton.

Definition 2.8 (Generated and marked languages) *The generated language of an automaton $G = (Q, \Sigma, f, q_0, Q_m)$ is*

$$\mathcal{L}(G) = \{s \in \Sigma^\star : f(q_0, s) \text{ is defined}\}.$$

The language marked by G is

$$\mathcal{L}_m(G) = \{s \in \mathcal{L}(G) : f(q_0, s) \in Q_m\}.$$

It is important to remark that, in Definition 2.8 the transition function is extended, *i.e.*, $f : Q \times \Sigma^\star \to Q$. Moreover, for any G such that $Q \neq \emptyset$, $\varepsilon \in \mathcal{L}(G)$.

The language $\mathcal{L}(G)$ is formed by all traces that can be created by following the transitions of the state transition diagram starting at the initial state. Therefore, a trace $s \in \mathcal{L}(G)$ if, and only if, it corresponds to an admissible path in the state transition diagram of G, *i.e.*, if, and only if, $f(q_0, s)$ is defined. It is important to remark that $\mathcal{L}(G)$ is prefix-closed by definition, since a path in G is only possible if all its prefixes are also possible. Moreover, if f is a total function over its domain, then $\mathcal{L}(G) = \Sigma^\star$. The generated language of an empty automaton, *i.e.*, an automaton whose state set is $Q = \emptyset$, is also the empty set. In this work, the generated language of G, $\mathcal{L}(G)$, is also referred as L, unless stated otherwise.

If $\Gamma_G(q) \neq \emptyset$ for all $q \in Q$, the language generated by $G = (Q, \Sigma, f, q_0, Q_m)$ is said to be live. The language marked by G, $\mathcal{L}_m(G)$, is a subset of L and represents all

traces s such that $f(q_0, s) \in Q_m$, *i.e.*, all traces that reach a marked state from the initial state q_0 in G. Notice that the language $\mathcal{L}_m(G)$ is not necessarily prefix-closed.

In the next section, we present some operations that can be applied to automata.

2.2.1 Operations on automata

There are basically two groups of operations that can be applied to automata: unary and composition operations [1].

Unary operations

The unary operations alter the state transition diagram of an automaton keeping its event set Σ unchanged. In the following, we present the definitions of the accessible and coaccessible part of an automaton.

Definition 2.9 (Accessible part) *The accessible part of an automaton G, $Ac(G)$, is defined as:*

$$Ac(G) = (Q_{ac}, \Sigma, f_{ac}, q_0, Q_{ac,m}),$$

where $Q_{ac} = \{q \in Q : (\exists s \in \Sigma^\star)[f(q_0, s) = q]\}$, $Q_{ac,m} = Q_m \cap Q_{ac}$, *and* $f_{ac} = f|_{Q_{ac}\times\Sigma\to Q_{ac}}$[1].

Taking the accessible part of an automaton G results in automaton $Ac(G)$, where all states of G, and its related transitions, that are not reachable from its initial state q_0 are deleted. It is important to notice that the accessible part operation does not change the generated and marked languages by G, $\mathcal{L}(G)$ and $\mathcal{L}_m(G)$.

A state $q \in Q$ is said to be coaccessible if there exists a path from the state q to a marked state. The Coaccessible operation erases all states of G, and their related transitions, that are not coaccessible. The formal definition of the coaccessible part of an automaton G is defined as follows [1].

Definition 2.10 (Coaccessible part) *The coaccessible part of an automaton G, $CoAc(G)$, is defined as:*

$$CoAc(G) = (Q_{coac}, \Sigma, f_{coac}, q_{0,coac}, Q_m),$$

where $Q_{coac} = \{q \in Q : (\exists s \in \Sigma^\star)[f(q, s) \in Q_m]\}$, $q_{0,coac} = q_0$ *if* $q_0 \in Q_{coac}$ *and* $q_{0,coac}$ *is not defined if* $q_0 \notin Q_{coac}$, *and* $f_{coac} = f|_{Q_{coac}\times\Sigma\to Q_{coac}}$.

[1]The notation $f|_{Q_{ac}\times\Sigma\to Q_{ac}}$ is used to indicate that we are restricting f to the smaller domain of the accessible states Q_{ac}.

Notice that taking the coaccessible part of G may shrink the generated language of G, *i.e.*, $\mathcal{L}(CoAc(G)) \subseteq \mathcal{L}(G)$. The marked language of G, $\mathcal{L}_m(G)$, is not altered when taking the coaccessible part of G, *i.e.*, $\mathcal{L}_m(CoAc(G)) = \mathcal{L}_m(G)$.

Composition operations

Composition operations are used to obtain a single automaton from two or more automata. In general, these operations are performed with the aim to construct a global system model from the automaton models of its components, modules or subsystems that operate concurrently. In this work, we define two composition operations: product and parallel composition [1].

The product composition, also known as completely synchronous composition, produces an automaton whose generated language is the intersection of the generated languages of the automata used in the composition. We formally define the product composition as follows.

Definition 2.11 (Product composition) *Let* $G_1 = (Q_1, \Sigma_1, f_1, q_{0,1})$ *and* $G_2 = (Q_2, \Sigma_2, f_2, q_{0,2})$ *be two automata. The product of* G_1 *and* G_2 *is the automaton*

$$G_1 \times G_2 = Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, f, (q_{0,1}, q_{0,2})),$$

where

$$f((q_1, q_2), \sigma) = \begin{cases} (f_1(q_1, \sigma), f_2(q_2, \sigma)) & \text{if } \sigma \in \Gamma_{G_1}(q_1) \cap \Gamma_{G_2}(q_2) \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

In the product, the transitions of the automata must always be synchronized on a common event, *i.e.*, in order to a transition (q, σ, q'), where $q = (q_1, q_2)$ and $q' = (q'_1, q'_2)$, belong to $G = G_1 \times G_2$, there must exist transitions (q_1, σ, q'_1) in G_1 and (q_2, σ, q'_2) in G_2 labeled with the same event σ. The product operation is restrictive, since it only allows transitions on common events. By definition, it can be verified that $\mathcal{L}(G_1 \times G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$. If $\Sigma_1 \cap \Sigma_2 = \emptyset$, then $\mathcal{L}(G_1 \times G_2) = \{\varepsilon\}$.

In general, systems are formed by simpler and smaller components or subsystems that interact and form the global system behavior. The component behavior can be classified into internal (private), and coupling behavior, that synchronizes with other components. These behaviors are modeled with private and common events, respectively. In order to model the global system behavior using the models of its components, there must exist an operation that is capable of integrating the system component models while taking into account their private behavior. This operation is called parallel composition and it is formally defined as follows.

Definition 2.12 (Parallel composition) *Let* $G_1 = (Q_1, \Sigma_1, f_1, q_{0,1})$ *and* $G_2 = (Q_2, \Sigma_2, f_2, q_{0,2})$ *be two automata. The parallel composition of* G_1 *and* G_2 *is the automaton*

$$G_1 \| G_2 = Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, f, (q_{0,1}, q_{0,2})),$$

where

$$f((q_1, q_2), \sigma) = \begin{cases} (f_1(q_1, \sigma), f_2(q_2, \sigma)) & \text{if } \sigma \in \Gamma_{G_1}(q_1) \cap \Gamma_{G_2}(q_2); \\ (f_1(q_1, \sigma), q_2) & \text{if } \sigma \in \Gamma_{G_1}(q_1) \setminus \Sigma_2; \\ (q_1, f_2(q_2, \sigma)) & \text{if } \sigma \in \Gamma_{G_2}(q_2) \setminus \Sigma_1; \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

In the parallel composition, a common event, *i.e.*, an event in $\Sigma_1 \cap \Sigma_2$, can only be executed in $G = G_1 \| G_2$ if it is executed by G_1 and G_2 simultaneously. The private events, *i.e.*, the events in $(\Sigma_1 \setminus \Sigma_2) \cup (\Sigma_2 \setminus \Sigma_1)$ can be executed whenever possible in G_1 and G_2. Thus, the parallel composition only synchronizes the common behavior of components, synchronizing their common events, and the private events (representing the private behavior of the components) can be executed whenever possible.

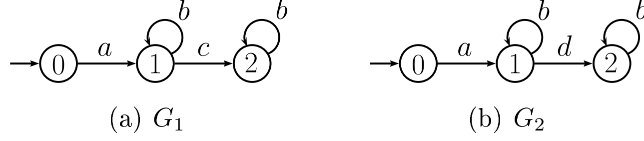
Let $P_i = (\Sigma_1 \cup \Sigma_2)^\star \rightarrow \Sigma_i^\star$ be two projections for $i = 1, 2$. The language generated by $G_1 \| G_2$ is equal to $\mathcal{L}(G_1 \| G_2) = P_1^{-1}(\mathcal{L}(G_1)) \cap P_2^{-1}(\mathcal{L}(G_2))$. If $\Sigma_1 = \Sigma_2$, then the parallel composition reduces to the product, and if $\Sigma_1 \cap \Sigma_2 = \emptyset$, then there are no synchronized transitions and $G_1 \| G_2$ models the concurrent behavior of G_1 and G_2.

In the following, we present an example of the product and parallel composition operations.

Example 2.2 *Let* $G_1 = (Q_1, \Sigma_1, f_1, q_{0,1})$ *and* $G_2 = (Q_2, \Sigma_2, f_2, q_{0,2})$ *be two automata, where* $\Sigma_2 = \{a, b, c\}$ *and* $\Sigma_2 = \{a, b, d\}$*, whose state transition diagrams are shown in Figure 2.2. In Figures 2.3(a) and 2.3(b) the automata* G_{prod} *and* G_{par}*, obtained by making the product and parallel compositions of automata* G_1 *and* G_2*, respectively, are presented. Notice that in automaton* G_{prod} *all transitions are labeled with events from* $\Sigma_1 \cap \Sigma_2 = \{a, b\}$*, while* G_{par} *models the synchronization of* G_1 *and* G_2*, through events* $\Sigma_1 \cap \Sigma_2 = \{a, b\}$*, and the concurrent behavior represented by the transitions labeled with events* c *and* d*.*

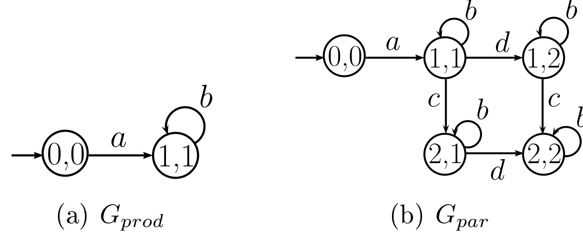
2.2.2 Automata with partially observed events

The set of events of an automaton Σ can be partitioned as $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, where Σ_o is the set of observable events and Σ_{uo} is the set of unobservable events. An event is observable when its occurrence can be registered by an external observer



(a) G_1 (b) G_2

Figure 2.2: Automata G_1 and G_2 of Example 2.2.



(a) G_{prod} (b) G_{par}

Figure 2.3: Automata G_{prod} and G_{par} of Example 2.2.

due, generally, to changes in sensors signals. Failure events, whose occurrence do not cause any immediate change in sensors readings, are modeled as unobservable events.

The observed language of a system G can be obtained from its generated language L by applying the projection $P_o(L)$, where $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$. In a system with unobservable events, it is important to know the set of possible states reachable from a given state $q \in Q$ after the occurrence of an unobservable event or traces formed by unobservable events. We call this set of states as unobservable reach, denoted by $UR(q)$, whose formal definition is presented as follows.

Definition 2.13 (Unobservable reach) *The unobservable reach of a state $q \in Q$, denoted by $UR(q)$, is defined as:*

$$UR(q) = \{y \in Q : (\exists t \in \Sigma_{uo}^\star)[f(q,t) = y]\}. \tag{2.2}$$

The unobservable reach can also be defined for a set of states $B \in 2^Q$ as:

$$UR(B) = \bigcup_{q \in B} UR(q). \tag{2.3}$$

The unobservable reach of a state q_ν is a set of states that corresponds to all states that are reached from q_ν by transitions labeled with unobservable events. The unobservable reach can be used to build an automaton from G that generates the observed language of G, $P_o(L)$. This automaton is called the observer of G, denoted by $Obs(G, \Sigma_o)$, and is defined as follows.

Definition 2.14 (Observer automaton) *The observer of an automaton G with respect to a set of observable events Σ_o, denoted by $Obs(G, \Sigma_o)$, is given by:*

$$Obs(G, \Sigma_o) = (Q_{obs}, \Sigma_o, f_{obs}, q_{0,obs}),$$

where $q_{obs} \subseteq 2^Q$. f_{obs}, and $q_{0,obs}$ are obtained following the steps of Algorithm 2.1 [1, 67].

Algorithm 2.1 *Observer automaton*

Input: *$G = (Q, \Sigma, f, q_0)$, and the observable event set Σ_o, where $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$.*
Output: *Observer automaton $Obs(G, \Sigma_o) = (Q_{obs}, \Sigma_o, f_{obs}, q_{0,obs})$.*

1: Define $q_{0,obs} \leftarrow UR(q_0)$. $Q_{obs} \leftarrow \{q_{0,obs}\}$ and $\widetilde{Q}_{obs} \leftarrow Q_{obs}$.

2: $\bar{Q}_{obs} \leftarrow \widetilde{Q}_{obs}$ and $\widetilde{Q}_{obs} \leftarrow \emptyset$.

3: For each $B \in \bar{Q}_{obs}$ do

3.1: $\Gamma_{obs}(B) \leftarrow \left(\bigcup_{q \in B} \Gamma_G(q)\right) \cap \Sigma_o$.

3.2: For each $\sigma \in \Gamma_{obs}(B)$,

$$f_{obs}(B, \sigma) \leftarrow UR(\{q \in Q : (\exists y \in B)[q = f(y, \sigma)]\}).$$

3.3: $\widetilde{Q}_{obs} \leftarrow \widetilde{Q}_{obs} \cup f_{obs}(B, \sigma)$.

4: $Q_{obs} \leftarrow Q_{obs} \cup \widetilde{Q}_{obs}$.

5: Repeat steps 2 to 4 until all accessible part of $Obs(G, \Sigma_o)$ is constructed.

In the following, we present an example of the observer of a system G.

Example 2.3 *Let G be the automaton whose transition state diagram is shown in Figure 2.4(a). The state set of G is $Q = \{0, 1, 2, 3\}$ and the event set of G is $\Sigma = \Sigma_{uo} \dot{\cup} \Sigma_o = \{a, b, \sigma_{uo}\}$, where $\Sigma_o = \{a, b\}$ and $\Sigma_{uo} = \{\sigma_{uo}\}$. The observer of G, $Obs(G, \Sigma_o)$, can be seen in Figure 2.4(b). If we assume that the system has executed trace $t = a\sigma_{uo}b$, then the observed trace is $P_o(t) = ab$, where $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$. It is important to notice that the state reached after the observation of trace $P_o(t) = ab$ in $Obs(G, \Sigma_o)$ is $\{2, 3\}$, which corresponds to the state estimate of G after the observation of trace t. Every state of $Obs(G, \Sigma_o)$ is a state estimate of G after the observation of a trace.*

In the next section, we present another mathematical formalism used in this work to represent DESs.

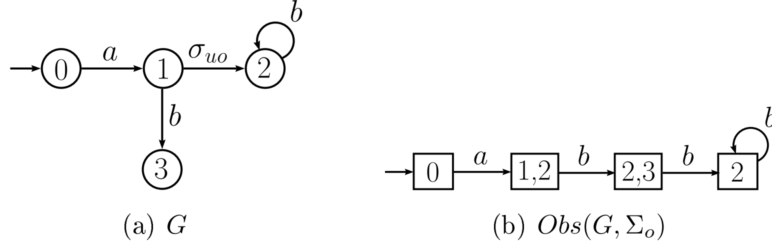


Figure 2.4: State transition diagram of automaton G of Example 2.3 (a), and observer automaton of G, $Obs(G, \Sigma_o)$, that provides the state estimates of G after the observation of a trace generated by the system (b).

2.3 Petri nets

A Petri net is a mathematical formalism used as an alternative to automata to represent DESs. Differently from automata, in a Petri net the state of the system is represented in a distributed way, which can be a better representation for concurrent and complex systems.

In a Petri net, events are associated with transitions and, in order to a transition occur, a set of conditions must be satisfied. The information related to these conditions is represented by the places of the net. Each transition has a set of input places that represent the conditions that have to be satisfied in order to the transition occur, and a set of output places that are related with the conditions that are affected by the transition occurrence.

2.3.1 Petri net structure

In a Petri net, there are two types of vertices: places and transitions. Places, transitions and the relations between then form the basic information that defines the structure of a Petri net. Each edge of the Petri net graph cannot connect vertices of the same type which makes the Petri net a bipartite graph. In the following, we present the formal definition of the structure of a Petri net [1, 4].

Definition 2.15 (Petri net structure) *The structure of a Petri net is a weighted bipartite graph*

$$(P, T, Pre, Post),$$

where P is the set of places, T is the set of transitions, $Pre : (P \times T) \to \mathbb{N} = \{0, 1, 2, \ldots\}$ is the function of arcs that connect places to transitions, and $Post : (T \times P) \to \mathbb{N}$ is the function of arcs that connect transitions to places.

The set of places is denoted by $P = \{p_1, p_2, \ldots, p_\nu\}$ and the set of transitions is denoted by $T = \{t_1, t_2, \ldots, t_\mu\}$. Therefore, $|P| = \nu$ and $|T| = \mu$, where $|.|$ denotes

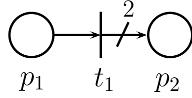


Figure 2.5: Structure of the Petri net of Example 2.4.

set cardinality. The set of input places (input transitions) of a transition $t_j \in T$ (place $p_i \in P$) is denoted by $I(t_j)$ ($I(p_i)$), and it is formed by the places $p_i \in P$ (transitions $t_j \in T$) such that $Pre(p_i, t_j) > 0$ ($Post(t_j, p_i) > 0$). Similarly, the set of output places (output transitions) of a transition $t_j \in T$ (place $p_i \in P$) is denoted by $Out(t_j)$ ($Out(p_i)$), and it is formed by the places $p_i \in P$ (transitions $t_j \in T$) such that $Post(t_j, p_i) > 0$ ($Pre(p_i, t_j) > 0$).

Graphically, places are represented by circles, while transitions are represented by bars. The functions *Pre* and *Post* determine the number of arcs that connect places to transitions and transitions to places. The value of the functions *Pre* and *Post* is represented only if it is different from 1. In the following, we present an example of a Petri net structure.

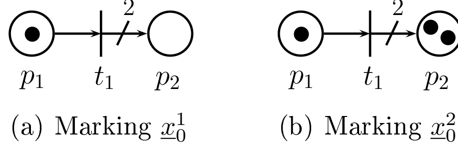
Example 2.4 *Let the structure of a Petri net, showed in Figure 2.5, be defined as* $P = \{p_1, p_2\}$, $T = \{t_1\}$, $Pre(p_1, t_1) = 1$, *and* $Post(t_1, p_2) = 2$. *In this example,* $I(t_1) = \{p_1\}$ *and* $I(p_2) = \{t_1\}$, $Out(p_1) = \{t_1\}$ *and* $Out(t_1) = \{p_2\}$.

2.3.2 Petri net marking

In a Petri net, the transitions are associated with events driving a DES, and places represent the conditions under which these transitions, and therefore the events associated with them, can occur. In this scheme, the element that indicate if these conditions are met is the assigning of tokens to places. The number of tokens assigned to a place is given by $x(p_i)$, where $x : P \to \mathbb{N}$ is a marking function. The marking of a Petri net is represented by the vector $\underline{x} = [x(p_1)\ x(p_2)\ \ldots\ x(p_\nu)]^T$, formed by the number of tokens assigned to each place p_i, for $i = 1, \ldots, \nu$. In the graphical representation of Petri nets, tokens are indicated by dark dots or numbers positioned in the appropriate places. We formally define Petri net as follows.

Definition 2.16 (Petri net) *A marked Petri net, or simply a Petri net* $\mathcal{N}$ *is a five-tuple* $\mathcal{N} = (P, T, Pre, Post, x_0)$, *where, according to Definition 2.15,* $(P, T, Pre, Post)$ *is the structure of the Petri net, and* x_0 *is the initial marking function of the Petri net.*

In a Petri net, the marking vector $\underline{x}$ represents the system state. For each new reachable state, the corresponding Petri net reaches a new marking. In the sequel, we present an example of a Petri net.



(a) Marking $\underline{x}_0^1$ (b) Marking $\underline{x}_0^2$

Figure 2.6: Two examples of Petri nets with different initial markings.

Example 2.5 *Consider again the Petri net structure of Example 2.4 depicted in Figure 2.5. In Figure 2.6 we show two possible initial markings* $\underline{x}_0^1 = [1\ 0]^T$ *and* $\underline{x}_0^2 = [1\ 2]^T$.

In a Petri net, a transition t_j is said to be enabled when the number of tokens assigned to each input place of t_j is greater or equal to the weight of the arcs that connect the places of $I(t_j)$ to transition t_j. The formal definition of an enabled transition is presented as follows.

Definition 2.17 (Enabled transition) *A transition* $t_j \in T$ *is said to be enabled if*

$$x(p_i) \geq Pre(p_i, t_j), \text{ for all } p_i \in I(t_j).$$

2.3.3 Petri net dynamics

In a Petri net, when a transition is enabled, it can fire, or occur. The state transition function of a Petri net is defined through the change in the marking of the places due to the firing of an enabled transition. If, for a given marking $\underline{x}$, an enabled transition t_j fires, the Petri net reaches a new marking $\underline{\overline{x}}$ given by

$$\overline{x}(p_i) = x(p_i) - Pre(p_i, t_j) + Post(t_j, p_i), \text{ for } i = 1, \ldots, \nu. \tag{2.4}$$

According to Equation (2.4), if p_i is an input place of t_j, and t_j fires, it loses a number of tokens equal to the weight of the arc that connects p_i to t_j, $Pre(p_i, t_j)$. If p_i is an output place of t_j, it gains as many tokens as the weight of the arc that connects t_j to p_i, $Post(t_j, p_i)$. Notice that p_i can be, at the same time, an input and output place of t_j. In this case, according to Equation (2.4), $Pre(p_i, t_j)$ tokens are removed from p_i and, at the same time, $Post(t_j, p_i)$ tokens are added to place p_i.

If, in a Petri net, a place p_i has at most one token, for all reachable markings of the net, then p_i is called safe. The Example 2.6 shows the firing of a transition and the evolution of the tokens resulting from it.

Example 2.6 *Consider the Petri net shown in Figure 2.7(a). Notice that transition* t_1 *is enabled for the marking* $\underline{x} = [1\ 0]^T$ *and, therefore,* t_1 *can fire. If* t_1 *fires, place*

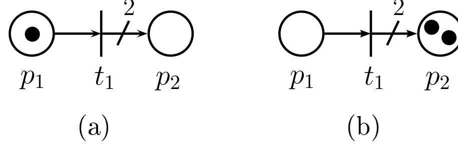


Figure 2.7: Petri net of Example 2.6 with transition t_1 enabled (a), and after the firing of transition t_1 with the new reached marking (b).

p_1 loses one token, since the weight of the arc that connects p_1 to t_1 is 1, and place p_2 receives two tokens, since the weight of the arc that connects t_1 to p_2 is 2. The firing of transition t_1 results in the marking $\overline{\underline{x}} = [0\ 2]^T$, depicted in Figure 2.7(b).

2.3.4 Labeled Petri net

In order to model DESs using the Petri net formalism, it is necessary to establish a correspondence between events and Petri net transitions. It is possible to use Petri nets to model DESs and represent languages if we associate at least one event to each transition of the net. This is carried out by a labeling function that associates a set of events to each transition. This leads to the following definition of a labeled Petri net.

Definition 2.18 (Labeled Petri net) *A labeled Petri net is a seven-tuple $\mathcal{N} = (P, T, Pre, Post, x_0, \Sigma, l)$, where $(P, T, Pre, Post, x_0)$ is, according to Definition 2.16, a Petri net. Σ is the set of events used to label transitions and $l : T \to 2^{\Sigma}$ is the transition labeling function that associates a subset of Σ to a transition in T.*

In a labeled Petri net, an enabled transition t_j fires when one of the events associated to t_j occurs. Example 2.7 illustrates a labeled Petri net.

Example 2.7 *Consider the labeled Petri net $\mathcal{N} = (P, T, Pre, Post, x_0, \Sigma, l)$ depicted in Figure 2.8, where $P = \{p_1, p_2\}$, $T = \{t_1, t_2, t_3\}$, $Pre(p_1, t_2) = Pre(p_2, t_3) = 1$, $Post(t_1, p_1) = Post(t_2, p_2) = 1$, $\underline{x}_0 = [0\ 1]^T$, $\Sigma = \{a, b, c\}$, $l(t_1) = \{a\}$, $l(t_2) = \{a, b\}$, and $l(t_3) = \{c\}$. Notice that transitions t_1 and t_3 are enabled and fire when events a or c occurs, respectively. It is important to notice that, if transition t_2 is enabled, it fires when event a or b occurs.*

2.3.5 State machine Petri net

A state machine Petri net (SMPN) is a special class of Petri nets where each transition has only one input place and one output place. Moreover, if this Petri net has only one token, then the SMPN has the same behavior as an automaton, where

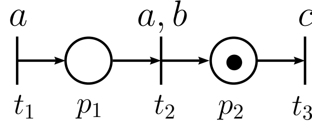


Figure 2.8: Labeled Petri net of Example 2.7.

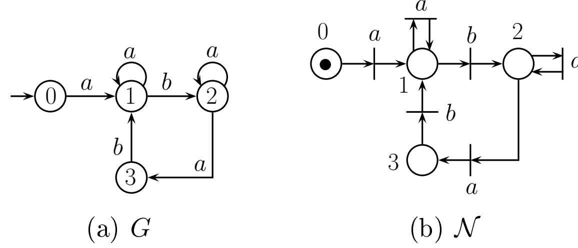


Figure 2.9: Automaton G (a), and state machine Petri net $\mathcal{N}$ (b) of Example 2.8.

each place is associated with a state of the corresponding automaton. Algorithm 2.2 illustrates the procedure for the construction of a SMPN from an automaton G.

Algorithm 2.2 *State machine Petri net*

Input: *Automaton* $G = (Q, \Sigma, f, q_0)$.
Output: *State machine Petri net* $\mathcal{N} = (P, T, Pre, Post, x_0, \Sigma, l)$.

1: *Create a place* $p_i \in P$ *associated with each state* $q_i \in Q$.

2: *Create a transition* $t_j \in T$ *for each transition* $q_\ell = f(q_i, \sigma)$ *defined in* G, *for all* $q_i \in Q$ *and* $\sigma \in \Gamma_G(q_i)$, *and define* $l(t_j) \leftarrow \{\sigma\}$.

3: *Define* $Pre(p_i, t_j) \leftarrow 1$ *and* $Post(t_j, p_\ell) \leftarrow 1$ *for each transition* $t_j \in T$, *if the transition* $q_\ell = f(q_i, \sigma)$ *is defined in* G. *Otherwise let* $Pre(p_i, t_j) \leftarrow 0$ *and* $Post(t_j, p_\ell) \leftarrow 0$.

4: *Make* $x_0(p_0) \leftarrow 1$ *and* $x_0(p_i) \leftarrow 0$ *for all* $p_i \in P \setminus \{p_0\}$, *where* p_0 *denotes the place associated with the initial state of* G, q_0.

Example 2.8 illustrates the equivalence between an automaton and its corresponding state machine Petri net.

Example 2.8 *Consider automaton* G *depicted in Figure 2.9(a). In Figure 2.9(b) we present the SMPN,* $\mathcal{N}$, *obtained from automaton* G *according to Algorithm 2.2. As it can be seen in Algorithm 2.2, in order to represent the exact behavior of an automaton using a SMPN, we have to replace the states of the automaton with places of the Petri net, and replace the arcs of the automaton with transitions of the Petri net, preserving the equivalence between the input and output transitions.*

2.3.6 Binary Petri net

Another class of Petri nets is the binary Petri net [68]. In a binary Petri net, the maximum number of tokens assigned to each place is forced to be one. Therefore, if a place that has one token and, after the firing of a transition, the same place receives another token, the place continues with only one token.

The binary Petri net can be defined as a Petri net with a different evolution rule for the marking of places after the firing of a transition t_j. This new evolution rule is defined as

$$\bar{x}(p_i) = \begin{cases} 0, \text{if } x(p_i) - Pre(p_i, t_j) + Post(t_j, p_i) = 0, \\ 1, \text{if } x(p_i) - Pre(p_i, t_j) + Post(t_j, p_i) > 0, \end{cases} \tag{2.5}$$

for $i = 1, \ldots, \nu$.

2.3.7 Extended Petri net

An extended Petri net is another class of Petri net that contains a special type of arc known as inhibitor arc [4]. An inhibitor arc is a direct arc that only connects places to transitions and its end is represented by a small circle. An extended labeled Petri net is defined as follows.

Definition 2.19 (Extended labeled Petri net) *An extended labeled Petri net is an eight-tuple $\mathcal{N} = (P, T, Pre, Post, In, x_0, \Sigma, l)$, where $(P, T, Pre, Post, x_0, \Sigma, l)$ is, according to Definition 2.18, a labeled Petri net, and $In : (P \times T) \to \mathbb{N}$ is the function of inhibitor arcs that only connects places to transitions.*

The inhibitor arc provides a new enabling rule to the transitions of the Petri net, such that if a place p_i is connected to a transition t_j by an inhibitor arc, transition t_j will be enabled if the number of tokens in p_i is smaller than the weight of the inhibitor arc that connects p_i to t_j, $In(p_i, t_j)$. The transition enabling rule in an extended Petri net is defined as follows.

Definition 2.20 (Enabled transition) *A transition $t_j \in T$ in an extended Petri net is said to be enabled if*

$$x(p_i) \geq Pre(p_i, t_j), \text{ and } x(p_i) < In(p_i, t_j), \text{ for all } p_i \in I(t_j),$$

where, now, we are considering that $p_i \in I(t_j)$, if $Pre(p_i, t_j) > 0$ or $In(p_i, t_j) > 0$.

Inhibitor arcs only enable or disable transitions, *i.e.*, if a transition t_j fires, where $I(p_i, t_j) > 0$, place p_i remains with the same number of tokens as before. The following example illustrates an extended labeled Petri net.

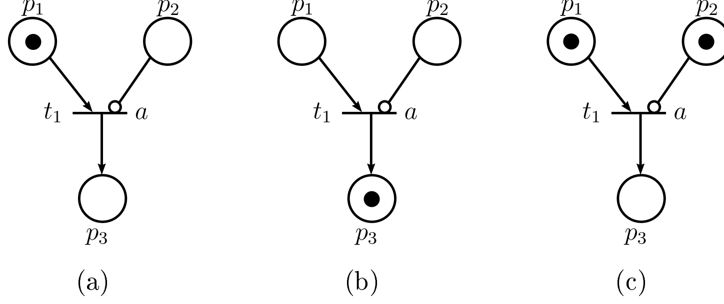


Figure 2.10: Petri net $\mathcal{N}$ of Example 2.9 with transition t_1 enabled (a), and after the firing of transition t_1 with the new reached marking (b). Petri net $\mathcal{N}$ with a different marking where transition t_1 is not enabled (c).

Example 2.9 *Consider the extended labeled Petri net* $\mathcal{N} = (P, T, Pre, Post, In, x_0, \Sigma, l)$ *shown in Figure 2.10(a), where* $P = \{p_1, p_2, p_3\}$, $T = \{t_1\}$, $Pre(p_1, t_1) = 1$, $Post(t_1, p_3) = 1$, $In(p_2, t_1) = 1$, $\underline{x}_0 = [1\ 0\ 0]^T$, $\Sigma = \{a\}$, $l(t_1) = \{a\}$. *Transition* t_1 *is enabled since, for the initial marking* $\underline{x}_0$, $x(p_1) \geq Pre(p_1, t_1)$ *and* $x(p_2) < In(p_2, t_1)$. *When event* a *occurs, transition* t_1 *fires and the extended Petri net reaches the new marking presented in Figure 2.10(b). Now, consider the same Petri net* $\mathcal{N}$ *with marking* $\underline{x}_2 = [1\ 1\ 0]^T$, *depicted in Figure 2.10(c). For this marking, transition* t_1 *is not enabled since* $x(p_2) = In(p_2, t_1) = 1$.

In the next section, we present the theoretical background of centralized, decentralized, and modular diagnosis of DESs modeled as automata.

2.4 Diagnosability of DESs

2.4.1 Centralized diagnosability of DESs

A common problem in DESs is to determine when a certain unobservable event, called failure event, has been executed by the system[2]. When this is possible, it is said that the system is diagnosable with respect to the projection $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$ and the failure event. Let G be the automaton that models a system and let $\mathcal{L}(G) = L$ be the language generated by G. Let $\Sigma_f \subseteq \Sigma_{uo}$ be the set of failure events, *i.e.*, the set of unobservable events whose occurrence must be diagnosed.

For the sake of simplicity, in this work it is assumed that there is only one failure event, *i.e.*, $\Sigma_f = \{\sigma_f\}$. There is no loss of generality in the results presented in this work by making this assumption since, for systems with more than one failure type, each failure type can be considered separately. In the sequel, we present the definition of nonfailure and failure traces of a system.

[2] In this work, a failure event is considered to be unobservable, since an observable failure event can be trivially diagnosed.

Definition 2.21 (Failure and nonfailure traces) *A failure trace is a trace of events s such that σ_f is one of the events that form s. A nonfailure trace, on the other hand, does not contain the event σ_f.*

The nonfailure language $L_N \subset L$ denotes the set of all nonfailure traces of L, and the subautomaton of G that generates L_N is denoted by G_N. Thus, the set of all traces generated by the system that contain σ_f is $L_F = L \setminus L_N$.

In SAMPATH *et al.* [14], the definition of language diagnosability is presented for systems that satisfy two assumptions:

A1. The language generated by the system is live;

A2. There is no cycle of unobservable events in the system.

Under these two assumptions, the following definition can be stated [14].

Definition 2.22 (Language diagnosability) *Let L and $L_N \subset L$ be the live and prefix-closed languages generated by G and G_N, respectively. Let $L_F = L \setminus L_N$. Then, L is said to be diagnosable with respect to projection $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$ and Σ_f if*

$$\begin{aligned}&(\exists z \in \mathbb{N})(\forall s \in L_F)(\forall st \in L_F)(\|t\| \geq z) \Rightarrow \\ &(P_o(st) \notin P_o(L_N)).\end{aligned}$$

According to Definition 2.22, L is diagnosable with respect to P_o and Σ_f if, and only if, for all failure traces st with arbitrarily long length after the occurrence of a failure event, there does not exist a nonfailure trace $s_N \in L_N$, such that $P_o(st) = P_o(s_N)$. Therefore, if L is diagnosable, then it is always possible to identify the occurrence of a failure event after a bounded number of observations of events.

In CASSANDRAS and LAFORTUNE [1], SAMPATH *et al.* [14, 15], a diagnoser automaton that can be used to verify the diagnosability of L and also for failure diagnosis is presented. This diagnoser is constructed based on an automaton G_l computed from the plant model G, where G_l is obtained by labeling the states of G according to the traces generated by the system, such that if a state of G is reached by a trace that contains the failure event σ_f, then it is labeled with F, otherwise it is labeled with N. After G_l has been obtained, the diagnoser automaton G_d is computed by making the observer of G_l with respect to its observable events, $G_d = Obs(G_l, \Sigma_o)$. The diagnoser automaton G_d is formally defined as follows.

Definition 2.23 (Diagnoser automaton) *The diagnoser automaton G_d obtained for the system G with respect to the failure set Σ_f and observable events set Σ_o is given by:*

$$G_d = (Q_d, \Sigma_o, f_d, q_{0,d}),$$

where $Q_d \subseteq 2^{Q \times \{N,F\}}$. *The transition function* f_d, *and the initial state* $q_{0,d}$ *are defined according to Algorithm 2.3.*

Algorithm 2.3 *Diagnoser automaton* G_d

Input: $G = (Q, \Sigma, f, q_0)$.

Output: *Diagnoser automaton* $G_d = (Q_d, \Sigma_o, f_d, q_{0,d})$.

1: *Define automaton* $A_l = (Q_l, \Sigma_f, f_l, q_{0,l})$, *where* $Q_l = \{N, F\}$, $f_l(N, \sigma_f) = F$, $f_l(F, \sigma_f) = F$, *and* $q_{0,l} = N$.

2: *Compute automaton* $G_l = G \| A_l$.

3: *Compute the diagnoser automaton* $G_d = Obs(G_l, \Sigma_o)$.

It is important to notice that automaton G_l generates the same language as automaton G. Moreover, the states of G_l are of the form $q_l = (q, N)$, such that $q \in Q$, if q is reached by a nonfailure trace, and $q_l = (q, F)$ if q is reached by a failure trace. The generated language of G_d is the natural projection of the generated language of G, L, *i.e.*, $\mathcal{L}(G_d) = P_o(L)$.

Since G_d is constructed from the observer automaton of G_l, the states of G_d are state estimates of G_l after the observation of a trace. If G_d reaches a state labeled only with the label F, the failure event has certainly occurred and it is diagnosed. A state of G_d labeled only with N indicates that the failure has not been executed by the system. States of G_d that have the labels N and F are called uncertain states, indicating that after the observation of a trace, a failure trace or a nonfailure trace with the same projection has been executed by the system.

In order to use G_d to verify the diagnosability of L, it is necessary to search for indeterminate cycles in G_d. An indeterminate cycle is an uncertain cycle, *i.e.*, a cycle formed by uncertain states, that is associated with at least two cycles in G_l, one that has only states labeled with N, and one that has only states labeled with F. If there is an indeterminate cycle in G_d, then the language generated by G, L, is not diagnosable, otherwise, L is diagnosable.

The following example illustrates the construction of the diagnoser automaton G_d for a given plant G. The state transition diagram of automaton A_l is also presented.

Example 2.10 *Consider the system* G *presented in Figure 2.11(a), such that* $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo} = \{a, b, c, \sigma_u, \sigma_f\}$, *where* $\Sigma_o = \{a, b, c\}$ *and* $\Sigma_{uo} = \{\sigma_u, \sigma_f\}$. *The failure event set is* $\Sigma_f = \{\sigma_f\}$. *The first step, according to Algorithm 2.3, to construct the diagnoser of* G, G_d, *is to build automaton* A_l, *whose state transition diagram is presented in Figure 2.12. Automaton* $G_l = G \| A_l$ *is depicted in Figure 2.11(b).*

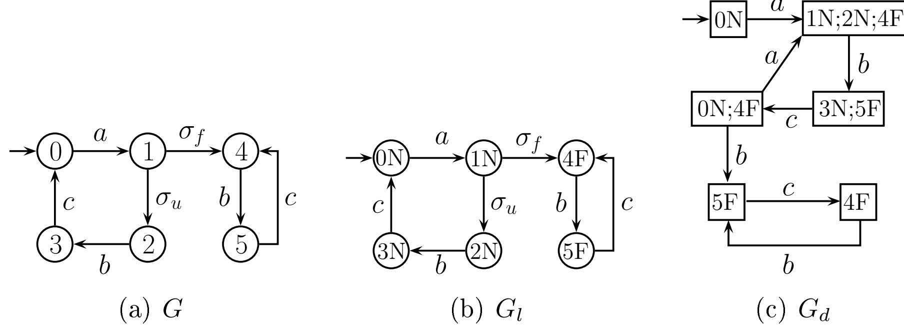


Figure 2.11: Automaton G (a), automaton G_l (b), and diagnoser automaton G_d (c) of Example 2.10.

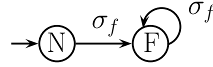


Figure 2.12: Automaton A_l of Example 2.10.

Finally, diagnoser G_d is obtained by computing the observer of G_l with respect to its observable event set Σ_o, $G_d = Obs(G_l, \Sigma_o)$, depicted in Figure 2.11(c).

Notice that the initial state of G_d is $\{0N\}$ which corresponds to the unobservable reach of the initial state of G_l. After the occurrence of event a, G_d reaches state $\{1N; 2N; 4F\}$. The fact that the labels N and F exist in state $\{1N; 2N; 4F\}$ indicates that, at this point, the diagnoser of G is not sure about the occurrence of the failure event. This also happens for states $\{3N; 5F\}$ and $\{0N; 4F\}$ in G_d after the observations of traces ab and abc, respectively. Notice that there exists an uncertain cycle in G_d formed by the states $\{1N; 2N; 4F\}$, $\{3N; 5F\}$, and $\{0N; 4F\}$. However, this cycle is associated only with states of G_l that have the label N and, thus, this cycle is not indeterminate. If the system executes the failure trace $a\sigma_f(bc)^z$ the failure event is diagnosed when G_d reaches state $\{5F\}$. Notice that, since there are no indeterminate cycles in G_d, the language of G is diagnosable with respect to $P_o : \Sigma^\star \to \Sigma_o^\star$ and Σ_f.

Now, let us consider that the observable event set of the system is $\Sigma'_o = \{b, c\}$, thus $\Sigma'_{uo} = \{a, \sigma_u, \sigma_f\}$. The diagnoser of G considering Σ'_o as the set of observable events, G'_d is shown in Figure 2.13. Notice that, there exists an uncertain cycle in G'_d formed by the states $\{0N; 1N; 2N; 4F\}$ and $\{3N; 5F\}$. This cycle is indeterminate since it is associated with the cycles in G_l labeled with N and F, namely the cycle formed by the states $\{0N\}$, $\{1N\}$, $\{2N\}$ and $\{3N\}$, and the cycle formed by the states $\{4F\}$ and $\{5F\}$.

Since the diagnoser automaton G_d is computed based on an observer, in the worst case, its state space can grow exponentially with the state space cardinality of the system $|Q|$. Therefore, its construction for diagnosability analysis is, in general,

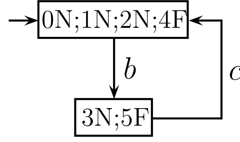


Figure 2.13: Diagnoser automaton G'_d considering Σ'_o as the set of observable events of Example 2.10.

avoided. Other algorithms developed exclusively to verify the diagnosability of L can also be found in the literature [69–71]. In these works, verifier automata, whose state space grows polynomially with the state space of the plant, are presented.

Petri net diagnoser

In SAMPATH *et al.* [14], it is stated that failure diagnosis can be carried out storing only the current state of the diagnoser, without the need for storing the complete state space of the diagnoser, and, after the observation of an event, the state estimate is updated. However, a method for this implementation is not presented in SAMPATH *et al.* [14]. In CABRAL *et al.* [26], a Petri net diagnoser (PND) for failure diagnosis of systems modeled as automata is proposed. The Petri net formalism is used to structure the diagnoser implementation, which is also presented in CABRAL *et al.* [26], where methods for the conversion of the PND into Programmable Logic Controller (PLC) programming languages are also proposed.

If language L is diagnosable with respect to P_o and Σ_f, then, the PND can be built in order to perform the diagnosis of the failure event. The PND is constructed from a state observer Petri net $\mathcal{N}_{SO}$, whose marking, after the observation of a trace, corresponds to the state estimate of the nonfailure behavior of the global system, G_N. Thus, in order to compute the PND for a system G, it is necessary to obtain automaton G_N whose generated language is the nonfailure language of the system L_N, where $L_N = L \setminus L_F$. Automaton G_N can be constructed following the steps of Algorithm 2.4 [71].

Algorithm 2.4 *Nonfailure model of the system.*

Input: *System model* $G = (Q, \Sigma, f, q_0)$*, and set of failure events* Σ_f*.*
Output: *Automaton* G_N*.*

1: Define $\Sigma_N \leftarrow \Sigma \setminus \Sigma_f$*.*

2: Build automaton A_N *composed of a single state* N*, that is also its initial state, with a self-loop labeled with all events in* Σ_N*.*

3: Construct the nonfailure automaton $G_N = G \times A_N = (Q_N, \Sigma, f_N, q_{0,N})$*.*

4: Redefine the event set of G_N as Σ_N, i.e., $G_N = (Q_N, \Sigma_N, f_N, q_{0,N})$.

After G_N has been computed, the state observer Petri net $\mathcal{N}_{SO}$ can be obtained by following the steps of Algorithm 2.5.

Algorithm 2.5 *State observer Petri net.*

Input: *Nonfailure system model* $G_N = (Q_N, \Sigma_N, f_N, q_{0,N})$.
Output: *State observer Petri net* $\mathcal{N}_{SO} = (P, T_{SO}, Pre_{SO}, Post_{SO}, x_{0,SO}, \Sigma_o, l_{SO})$.

1: *Compute the SMPN* $\mathcal{N} = (P, T, Pre, Post, x_0, \Sigma, l)$ *from* G_N *by using Algorithm 2.2.*

2: *Define the function* $Reach_T(t_j)$, $Reach_T : T_o \to 2^P$, *where* $t_j \in T_o$, *and* T_o *is the set of all transitions of* $\mathcal{N}$ *labeled with observable events, as follows:*

   2.1: *Let* $Out(P) \leftarrow \cup_{p \in P} Out(p)$ *and* $Out(T) \leftarrow \cup_{t \in T} Out(t)$.

   2.2: *Let* $\{p_{out}\} \leftarrow Out(t_j)$, $P'_r \leftarrow \{p_{out}\}$, *and* $P_r \leftarrow P'_r$.

   2.3: *Let* T'_u *be the set of all transitions of* $Out(P'_r)$ *labeled with unobservable events. If* $T'_u = \emptyset$, $Reach_T(t_j) \leftarrow P_r$ *and stops.*

   2.4: *Set* $P'_r \leftarrow Out(T'_u)$, $P_r \leftarrow P_r \cup P'_r$, *and return to Step 2.3.*

3: *Add to* $\mathcal{N}$ *arcs connecting each observable transition* $t_j \in T_o$ *to the places in* $Reach_T(t_j)$, *generating the Petri net* $\mathcal{N}' = (P, T, Pre, Post', x_0, \Sigma, l)$.

4: *Eliminate all transitions of* $\mathcal{N}'$ *labeled with unobservable events and their related arcs, generating the binary Petri net* $\mathcal{N}_o = (P, T_o, Pre_o, Post_o, x_0, \Sigma_o, l_o)$.

5: *Compute* $\mathcal{N}_{SO} = (P, T_{SO}, Pre_{SO}, Post_{SO}, x_{0,SO}, \Sigma_o, l_{SO})$ *as follows:*

   5.1: *Set* $T'_o \leftarrow \emptyset$. *For all* $q_{N_i} \in Q_N$ *such that* $\Gamma_{G_N}(q_{N_i}) \cap \Sigma_o \neq \Sigma_o$, *create a new transition* t^i *and let* $T'_o \leftarrow T'_o \cup \{t^i\}$.

   5.2: *Set* $T_{SO} \leftarrow T_o \cup T'_o$.

   5.3: *Define the new labeling function* $l_{SO} : T_{SO} \to 2^{\Sigma_o}$, *where* $l_{SO}(t_j) \leftarrow l_o(t_j)$, *if* $t_j \in T_o$, *and* $l_{SO}(t^i) \leftarrow \Sigma_o \setminus (\Gamma_{G_N}(q_{N_i}) \cap \Sigma_o)$, *if* $t^i \in T'_o$.

   5.4: *Define* $Pre_{SO} : P \times T_{SO} \to \mathbb{N}$ *and* $Post_{SO} : T_{SO} \times P \to \mathbb{N}$, *where* $Pre_{SO}(p_i, t_j) \leftarrow Pre_o(p_i, t_j)$ *and* $Post_{SO}(t_j, p_\ell) \leftarrow Post_o(t_j, p_\ell)$, *for all* $p_i, p_\ell \in P$ *and* $t_j \in T_o$, *and* $Pre_{SO}(p_i, t^i) \leftarrow 1$, $Pre_{SO}(p_\ell, t^i) \leftarrow 0$ *and* $Post_{SO}(t^i, p_\ell) \leftarrow 0$ *and* $Post_{SO}(t^i, p_i) \leftarrow 0$, *for all* $t^i \in T'_o$ *and* $p_i, p_\ell \in P$ *where* $i \neq \ell$.

   5.5: *Define the initial state of* $\mathcal{N}_{SO}$ *by assigning a token to each place associated with a state of* $UR(q_{0,N})$ *and zero to the other places.*

5.6: Redefine T_{SO}, Pre_{SO}, and $Post_{SO}$ by eliminating the self-loop transitions and their associated arcs.

In the Petri net state observer $\mathcal{N}_{SO}$, the places that have tokens after the observation of a trace correspond to the state estimate of G_N. Assume that the language L is diagnosable with respect to P_o and Σ_f, then, according to Definition 2.22, after a bounded number of occurrences of events after the failure event, all places of the Petri net $\mathcal{N}_{SO}$ will have zero tokens. In order to use the Petri net $\mathcal{N}_{SO}$ for diagnosis, a failure detection logic that indicates the failure occurrence when all places of $\mathcal{N}_{SO}$ have zero tokens must be added to $\mathcal{N}_{SO}$. The addition of the failure detection logic to $\mathcal{N}_{SO}$ leads to the Petri net diagnoser $\mathcal{N}_D$ that can be constructed according to Algorithm 2.6.

Algorithm 2.6 *Petri net diagnoser.*

Input: *Petri net state observer* $\mathcal{N}_{SO} = (P, T_{SO}, Pre_{SO}, Post_{SO}, x_{0,SO}, \Sigma_o, l_{SO})$.
Output: *Petri net diagnoser* $\mathcal{N}_D = (P_D, T_{SO} \cup t_f, Pre_D, Post_D, In_D, x_{0,D}, \Sigma_o \cup \{\lambda\}, l_D)$.

1: Let t_f be a transition created to identify the occurrence of a failure event of the set Σ_f. $T_D \leftarrow T_{SO} \cup t_f$.

2: Define the labeling function $l_D : T_D \to 2^{\Sigma_o \cup \{\lambda\}}$, where λ is the always occurring event, such that $l_D(t_D) \leftarrow l_{SO}(t_D)$ for all $t_D \in T_{SO}$, and $l_D(t_f) \leftarrow \{\lambda\}$.

3: Add to transition t_f an input place p_N and an output place p_F. $P_D \leftarrow P \cup \{p_N, p_F\}$.

4: Define $Pre_D : P_D \times T_D \to \mathbb{N}$ and $Post_D : T_D \times P_D \to \mathbb{N}$ where $Pre_D(p_i, t_D) \leftarrow Pre_{SO}(p_i, t_D)$ and $Post_D(t_D, p_i) \leftarrow Post_{SO}(t_D, p_i)$ for all $t_D \in T_{SO}$ and $p_i \in P$, $Pre_D(p_N, t_f) \leftarrow 1$ and $Post_D(t_f, p_F) \leftarrow 1$, and $Pre_D(p_i, t_f) \leftarrow 0$ and $Post_D(t_f, p_i) \leftarrow 0$ for all $p_i \in P$.

5: Define the function of inhibitor arcs $In_D : P_D \times T_D \to \{0, 1\}$, where $In_D(p_D, t_f) = 1$ for all $p_D \in P$, and $In_D(p_D, t_D) = 0$ for all other places $p_D \in P_D$ and transitions $t_D \in T_D$.

6: The initial marking of place p_N is one and of place p_F is zero. The other places have the same initial marking defined by $x_{0,SO}$.

The PND $\mathcal{N}_D$ computed from Algorithm 2.6 has polynomial growth with the size of the plant model G [26]. Methods for the conversion of the PND into ladder diagram and sequential function chart in order to be implemented in a programmable

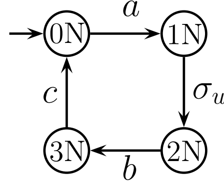


Figure 2.14: Nonfailure behavior automaton G_N of the system G of Example 2.11.

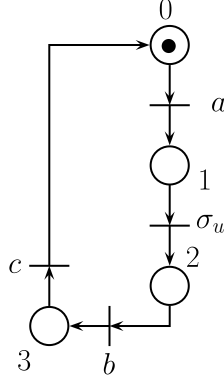


Figure 2.15: State machine Petri net $\mathcal{N}$ of Example 2.11.

logic controller are also presented in CABRAL *et al.* [26]. In the following example, we illustrate the construction of the PND $\mathcal{N}_D$ and the diagnosis method for a system G.

Example 2.11 *Consider the plant model G of Example 2.10 depicted in Figure 2.11(a). In order to construct the Petri net diagnoser $\mathcal{N}_D$, it is necessary first to obtain the nonfailure automaton G_N according to Algorithm 2.4. Automaton G_N is depicted in Figure 2.14. Once G_N has been computed, the state observer Petri net $\mathcal{N}_{SO}$ can be obtained according to Algorithm 2.5. Following Algorithm 2.5, the first step to construct $\mathcal{N}_{SO}$ is to compute the state machine Petri net $\mathcal{N}$ shown in Figure 2.15. The state observer Petri net $\mathcal{N}_{SO}$ obtained according to Algorithm 2.5 is depicted in Figure 2.16. Following Algorithm 2.6, the Petri net diagnoser $\mathcal{N}_D$, presented in Figure 2.17 is constructed. Now, consider that the system executes trace $s = a\sigma_f bcb$. When event a is observed, transition t_2 of $\mathcal{N}_D$ will fire, removing the token from place 0 and adding one token to places 1 and 2. When event b is observed, transitions t_3 and t_5 fire, which remove the tokens from places 1 and 2, and add a token to place 3. When event c is observed, transition t_7 fires and the initial marking of $\mathcal{N}_D$ is reached again. Finally, when the second occurrence of event b is observed, transition t_1 fires, removing the token from place 0. At this moment, transition t_f is enabled and, since it is labeled with the always occurring event λ, it fires, removing a token from p_N and adding a token to p_F, diagnosing the failure event occurrence.*

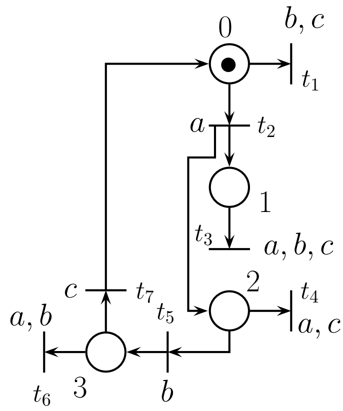


Figure 2.16: State observer Petri net $\mathcal{N}_{SO}$ of Example 2.11.

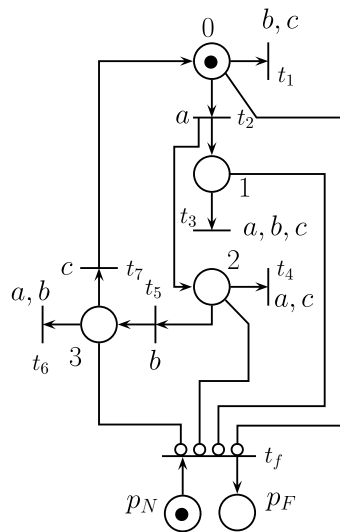


Figure 2.17: Petri net diagnoser $\mathcal{N}_D$ of Example 2.11.

2.4.2 Codiagnosability of DESs

In this work, we consider the decentralized diagnosis scheme as described in Protocol 3 of DEBOUK *et al.* [17]. The Protocol 3 of DEBOUK *et al.* [17] consists of ℓ local diagnosers that do not communicate among each other, where each local diagnoser has its own set of observable events. Thus, for each local diagnoser, the set of events can be partitioned as $\Sigma = \Sigma_{o_i}\dot{\cup}\Sigma_{uo_i}$, for $i = 1, \ldots, \ell$. In this scheme, the failure is diagnosed when at least one of the local diagnosers identifies its occurrence, and the diagnosis decision is sent to a coordinator. It is important to remark that in the decentralized diagnosis scheme proposed in DEBOUK *et al.* [17], two different observable event sets may have events in common, *i.e.*, $\Sigma_{o_i} \cap \Sigma_{o_j}$ is not necessarily equal to the empty set, for $i \neq j$, $i, j \in \{1, \ldots, \ell\}$. In addition, it is also assumed in DEBOUK *et al.* [17] that the language of the system is live.

The following definition of language codiagnosability can be stated [17].

Definition 2.24 (Language codiagnosability) *Let L be the live language generated by G. Then, L is said to be disjunctively codiagnosable with respect to projections $P_{o_i} : \Sigma^\star \rightarrow \Sigma_{o_i}^\star$, for $i = 1, \ldots, \ell$, and Σ_f if*

$$(\exists z \in \mathbb{N})(\forall s \in L \setminus L_N)(\forall st \in L \setminus L_N, \|t\| \geq z) \Rightarrow$$
$$(\exists i \in \{1, \ldots, \ell\})[P_{o_i}(st) \notin P_{o_i}(L_N)].$$

According to Definition 2.24, L is codiagnosable with respect to P_{o_i} and Σ_f if, and only if, for all failure traces st with arbitrarily long length after the occurrence of a failure event, there do not exist nonfailure traces $\omega_i \in L_N$, such that $P_{o_i}(st) = P_{o_i}(\omega_i)$ for all $i \in \{1, \ldots, \ell\}$. Therefore, if L is codiagnosable, then it is always possible to identify the occurrence of a failure event after a bounded number of event observations. Notice that the diagnosability definition 2.22 can be obtained from Definition 2.24 by making $\ell = 1$.

In order to implement a decentralized diagnosis scheme, it is first necessary to verify if the system is codiagnosable, *i.e.*, verify if it is always possible to identify if a failure has occurred after a finite number of event observations after the occurrence of the failure event. In MOREIRA *et al.* [71], a polynomial-time algorithm is presented to verify if the language L is codiagnosable with respect to $P_{o_i} : \Sigma^\star \rightarrow \Sigma_{o_i}^\star$, for $i = 1, \ldots, \ell$, and Σ_f. In the sequel, we present the verifier algorithm presented in MOREIRA *et al.* [71].

Algorithm 2.7 *Codiagnosability verification.*

Input: *System model $G = (Q, \Sigma, f, q_0)$, set of failure events Σ_f, and $\Sigma = \Sigma_{o_i}\dot{\cup}\Sigma_{uo_i}$, $i = 1, \ldots, \ell$.*

Output: *Codiagnosability decision.*

1: *Compute automaton G_N by following the steps of Algorithm 2.4.*

2: *Compute automaton G_F, whose marked language corresponds to the failure behavior of the system, as follows:*

2.1: *Set $A_l = (Q_l, \Sigma_f, f_l, q_{0,l})$, where $Q_l = \{N, F\}$, $q_{0,l} = \{N\}$, $f_l(N, \sigma_f) = F$ and $f_l(F, \sigma_f) = F$, for all $\sigma_f \in \Sigma_f$.*

2.2: *Compute $G_l = G \| A_l$ and mark all states of G_l whose second coordinate is equal to F.*

2.3: *Compute the failure automaton $G_F = CoAc(G_l)$.*

3: *Define the function $R_i : \Sigma_N \to \Sigma_{R_i}$ as:*

$$R_i(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_{o_i} \\ \sigma_{R_i}, & \text{if } \sigma \in \Sigma_{uo_i} \setminus \Sigma_f \end{cases} . \tag{2.6}$$

Construct automata $G_{N,i} = (Q_N, \Sigma_{R_i}, f_{N,i}, q_{0,N})$, for $i = 1, \ldots, \ell$, with $f_{N,i}(q_N, R_i(\sigma)) = f_N(q_N, \sigma)$ for all $\sigma \in \Sigma_N$.

4: *Compute the verifier automaton $G_V = (\|_{i=1}^{\ell} G_{N,i}) \| G_F = (Q_V, (\cup_{i=1}^{\ell} \Sigma_{R_i}) \cup \Sigma, f_V, q_{0,V})$.*

5: *Verify the existence of a cycle $cl = (q_V^{\delta}, \sigma_{\delta}, q_V^{\delta+1}, \ldots, q_V^{\gamma}, \sigma_{\gamma}, q_V^{\delta})$, where $\gamma \geq \delta > 0$, in G_V satisfying the following conditions:*

$$\exists j \in \{\delta, \delta+1, \ldots, \gamma\} \text{ s.t. for some } q_V^j, \left(q_l^j = F\right) \wedge (\sigma_j \in \Sigma) . \tag{2.7}$$

If the answer is yes, then L is not codiagnosable with respect to P_{o_i} and Σ_f. Otherwise, L is codiagnosable.

Notice that a state of G_V is given by $q_V = (q_{N,1}, q_{N,2}, \ldots, q_{N,\ell}, q_F)$ where $q_{N,1}, q_{N,2}, \ldots, q_{N,\ell}$, and q_F are the states of $G_{N,1}$, $G_{N,2}$, $\ldots$, $G_{N,\ell}$, and G_F, respectively, and $q_F = (q, q_l)$, where q and q_l are states of G and A_l, respectively. Algorithm 2.7 can be used to verify the centralized diagnosability of L by making $\ell = 1$, *i.e.*, by considering only one diagnoser. It is also important to remark that assumptions A1 and A2 are removed when verifiers are used instead of diagnosers [19, 71]. In the sequel, we present an example to illustrate the use of Algorithm 2.12 for the verification of the codiagnosability.

Example 2.12 *Consider the system G depicted in Figure 2.18 and suppose we want to verify the codiagnosability of L with respect to $P_{o_i} : \Sigma^{\star} \to \Sigma_{o_i}^{\star}$, $i = 1, 2$ and*

Σ_f, where $\Sigma = \{a, b, c, \sigma_f\}$, $\Sigma_{o_1} = \{a, c\}, \Sigma_{o_2} = \{b, c\}$, and $\Sigma_f = \{\sigma_f\}$. In steps 1 and 2, automata G_N and G_F presented in Figure 2.19(a) and (b), respectively, are computed. In the sequel, automata $G_{N,1}$ and $G_{N,2}$ are built in Step 3. In this example, automata $G_{N,1}$ and $G_{N,2}$ are equal to automaton G_N and, thus, are omitted. Finally, the verifier automaton G_V is shown in Figure 2.20. Notice that there are no cycles in G_V satisfying conditions (2.7). Therefore, the language generated by G is codiagnosable with respect to P_{o_i} and Σ_f.

If the language L is codiagnosable with respect to P_{o_i} and Σ_f, a decentralized diagnosis scheme can be implemented with local diagnosers. The Petri net diagnoser presented in Section 2.4.1 can be used to perform decentralized diagnosis. In order to do so, it is necessary to build local Petri net state observers $\mathcal{N}_{SO_i}$ for each site considering its own set of observable events Σ_{o_i} in Algorithm 2.5. After the Petri nets $\mathcal{N}_{SO_i}$ are computed, for $i = 1, \ldots, \ell$, a failure detection logic must be added to $\mathcal{N}_{SO_i}$, according to Algorithm 2.6, generating the local Petri net diagnosers $\mathcal{N}_{D_i}$. Each local diagnoser $\mathcal{N}_{D_i}$ will have its own failure place p_{F_i}, whose marking must be communicated to a coordinator in order to inform the diagnosis of the failure event.

In the sequel, we present another diagnosis framework, known as modular diagnosis [59]. The idea in this architecture is to avoid the use of the global plant model for diagnosis, using only a local diagnoser for the failure component of the system.

2.4.3 Modular diagnosability of DESs

Different modular diagnosis approaches have been presented in the literature [56, 58–60]. In PENCOLÉ and CORDIER [58], a local diagnoser is computed for each component of the system and the diagnoses are merged in order to obtain the global diagnosis decision. The main drawback of the work presented in PENCOLÉ and CORDIER [58] is that, in the worst-case, the paths of all modules of the system must be synchronized, which leads to an exponential growth with the number of system components. In ZHOU *et al.* [60], a decentralized modular diagnosis scheme for DESs is presented, where it is introduced the notion of local nonfailure specifications for modular diagnosability. In ZHOU *et al.* [60] it is stated that the local nonfailure specifications are not unique, and a method for the computation of these

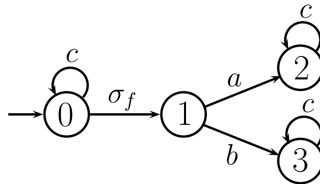


Figure 2.18: Automaton G of Example 2.12.

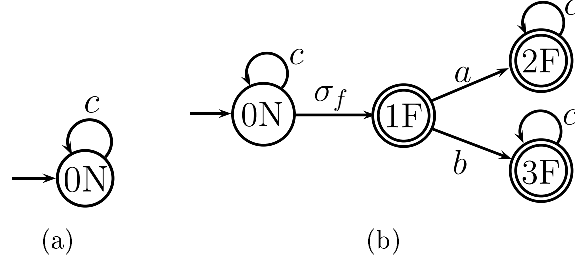


Figure 2.19: Automaton G_N (a) and automaton G_F (b) of Example 2.12.

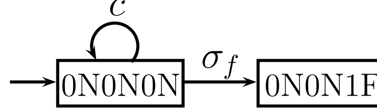


Figure 2.20: Automaton G_V of Example 2.12.

specifications is not presented. Since the modular architectures proposed in [58, 60] are incomparable with the notions of synchronous diagnosability and synchronous codiagnosability presented in this work we do not further exploit these approaches.

In DEBOUK *et al.* [56] a different modular architecture for DESs is proposed. The idea is to compute local diagnosers obtained by following the steps of Algorithm 2.3 for each component of the system. The local diagnoser only diagnoses the failure modeled in its respective module. In DEBOUK *et al.* [56], the classical definition of diagnosability [14] is used, and sufficient conditions that ensure global diagnosability using the modular architecture are proposed.

In CONTANT *et al.* [59], a notion of modular diagnosability that is different from the monolithic notion of diagnosability [14], is proposed. Necessary and sufficient conditions that ensure the modular diagnosability of a DES are presented. Similar to DEBOUK *et al.* [56], the modular diagnosis architecture proposed in CONTANT *et al.* [59] consists on the computation of local diagnosers that can infer the global occurrence of the failure event by observing only the local component model where the failure is modeled. Due to the local diagnoser implementation with the aim to diagnose a global failure occurrence by observing only the failure component model, the notion of modular diagnosability proposed in CONTANT *et al.* [59] can be compared to the notions of synchronous diagnosability and synchronous codiagnosability proposed in this work. In order to do so, in this section we present the modular diagnosis architecture, the notion of modular diagnosability and the assumptions introduced in CONTANT *et al.* [59] that ensure the necessary and sufficient conditions for modular diagnosability.

Let us consider that the global system model is obtained by the parallel composition of its subsystems or components, *i.e.*, $G = \|_{k=1}^{r} G_k$, where r is the total

number of system components. In order to introduce the definition of modular diagnosability, we first present the assumptions considered in CONTANT *et al.* [59].

- **A1.** The language of the system L is live, and there are no cycles of unobservable events in the system component models G_k, for $k = 1, \ldots, r$;
- **A2.** Common events between two or more components are observable;
- **A3.** The model that exhibits the failure behavior has persistent excitation, *i.e.*, the failure does not bring the system to a halt.

Based on Assumptions **A1**-**A3**, and considering that the system is formed by the composition of all modules G_k, $k = 1, \ldots, r$, and that the failure event is modeled only in automaton G_y, $y \in \{1, \ldots, r\}$, the following definition of modular diagnosability can be stated [59].

Definition 2.25 (Modular diagnosability) *Let* $G = \|_{k=1}^{r} G_k$*, and let* G_y*, for* $y \in \{1, \ldots, r\}$*, be the automaton that models the failure component. The language* $\mathcal{L}(G) = L$ *is said to be modularly diagnosable with respect to* $\Sigma_o = \cup_{k=1}^{r} \Sigma_{k,o}$ *and* $\Sigma_f \subseteq \Sigma_y$ *if*

$$(\exists z' \in \mathbb{N})(\forall s \in L_F)(\forall st \in L_F, \|P_{y,o}(t)\| \geq z') \Rightarrow$$
$$P_o(st) \notin P_o(L_N).$$

We now explain the effects of assumptions **A1-A3** of the modular diagnosability definition.

Assumption **A1** is considered in CONTANT *et al.* [59] in order to avoid the existence of hidden cycles [72] in the diagnosers, since the diagnosers are based on observers. Thus, observers can be used in order to verify the notion of modular diagnosability, as it is done in CONTANT *et al.* [59].

Assumption **A2** is stated in order to guarantee that if the language of the system L is diagnosable, then L is modularly diagnosable, and that if the language of the module G_y, $L_y = \mathcal{L}(G_y)$, for $y \in \{1, \ldots, r\}$ is diagnosable, then L is modularly diagnosable[3]. This result guarantees that, under Assumption **A2**, only the local diagnoser associated with module G_y can be used to perform online failure diagnosis of the system. It is important to remark that, based only on Assumption **A2**, if L_y and L are both nondiagnosable, then the modular diagnosability is not guaranteed.

Consider now the implications of Assumption **A3**. If persistent of excitation of the failure component is guaranteed, then there does not exist a failure trace with a suffix of arbitrarily long length formed only with events that belong to the

[3]In the next chapters we show why Assumption **A2** is needed to ensure this result.

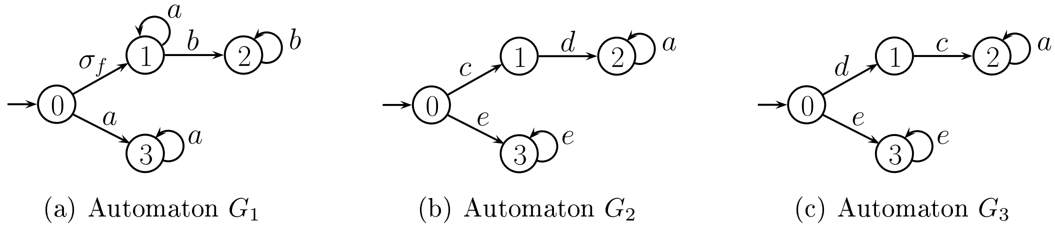
modules G_k, $k \in \{1, \ldots, r\}$ and $k \neq y$. In other words, there cannot exist an arbitrary long length failure trace in the global behavior model G such that its arbitrarily long length suffix is formed only with events that do not belong to the failure component. In practice, Assumption **A3** excludes traces from L_F that are known to be impossible to be executed by the system. This implies that the failure behavior of the system can be modeled by a reduced failure language $L_F^{red} \subset L_F$. Therefore, the modular diagnosability definition is equivalent, under Assumption **A3**, to a weaker definition of diagnosability, which can be explicitly considered in Definition 2.25 by replacing L_F with L_F^{red}.

It is also important to remark that, according to Assumptions **A1** and **A3**, there exists a suffix t associated with any trace $s \in L_F^{red}$, such that $st \in L_F^{red}$ and $P_{y,o}(t) \neq \varepsilon$. This fact, together with the fact that under Assumption **A2**, the diagnosis can be performed only by the local diagnoser associated with G_y, implies that condition $\|t\| \geq z$ can be replaced with $\|P_{y,o}(t)\| \geq z'$ in the classical definition of diagnosability (Definition 2.22), leading to the definition of modular diagnosability (Definition 2.25).

In the following, we present an example, also presented in CONTANT *et al.* [59], that shows the main differences between the notions of diagnosability and modular diagnosability of the language of the system L.

Example 2.13 *Let $G = G_1\|G_2\|G_3$, where automata G_1, G_2 and G_3 are depicted in Figure 2.21 and automaton G is presented in Figure 2.22. The set of events of G_1, G_2 and G_3 are $\Sigma_1 = \Sigma_{1,uo}\dot{\cup}\Sigma_{1,o} = \{a, b, \sigma_f\}$, $\Sigma_2 = \Sigma_{2,o} = \{a, c, d, e\}$, and $\Sigma_3 = \Sigma_{3,o} = \{a, c, d, e\}$, respectively, where $\Sigma_{1,uo} = \Sigma_f = \{\sigma_f\}$, $\Sigma_{1,o} = \{a, b\}$, $\Sigma_o = \Sigma_{1,o} \cup \Sigma_{2,o} \cup \Sigma_{3,o} = \{a, b, c, d, e\}$, and $\Sigma_{uo} = \{\sigma_f\}$. In order to investigate the modular diagnosability and monolithic diagnosability, we build the diagnoser automata of G_1 and G, G_{d_1} and G_d, depicted in Figures 2.23 and 2.24, respectively, according to Algorithm 2.3. Notice that there is an indeterminate cycle in the diagnoser automaton G_d. Considering only the system model automaton G, the indeterminate cycle in G_d indicates that the system is not monolithically diagnosable according to Definition 2.22.*

*Let us now analyze the modular diagnosability of L. Notice that, there is an indeterminate cycle in G_{d_1}, which would indicate that L is not modularly diagnosable. However, notice that the indeterminate cycle in G_{d_1} is not executed due to the interaction between modules G_1, G_2 and G_3. In other words, transitions $((0N; 1F), a, (1F; 3N))$ and $((1F; 3N), a, (1F; 3N))$ will not be executed in G_{d_1} since, event a will not be executed in module G_1 due to the interaction of G_1 with modules G_2 and G_3. Moreover, Assumption **A3** guarantees that module G_1, which has the failure event modeled, has persistent excitation. Thus, if the failure event σ_f occurs in the system, G_1 will execute event b due to Assumption **A3** and the occurrence of*



(a) Automaton G_1 (b) Automaton G_2 (c) Automaton G_3

Figure 2.21: Automata G_1, G_2 and G_3 of Example 2.13.

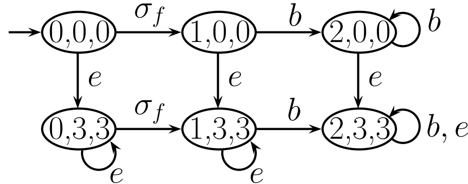


Figure 2.22: Automaton G of Example 2.13.

*σ_f would be diagnosed by diagnoser G_{d_1}. Notice that, as a consequence of Assumption **A3**, the system cannot generate the trace $\sigma_f e^z$ for an arbitrarily large value of z. Therefore, the language L is modularly diagnosable with respect to $\Sigma_o = \cup_{k=1}^{r} \Sigma_{k,o}$ and $\Sigma_f = \{\sigma_f\}$, and a local diagnoser can be constructed based only in module G_1 in order to diagnose the failure event occurrence.*

2.5 Final remarks

In this chapter, the formal definition of the language of a DES and two formalisms capable of representing DESs behavior were presented: (i) automata, and (ii) Petri nets. In this work, DESs are modeled as automata, which makes the unary and the composition operations fundamental tools in order to analyze and construct models of complex DESs from simple models of components or subsystems. Although the systems considered in this work are modeled using automata, the Petri net formalism is used to synthesize the diagnosers proposed in this work. The Petri net formalism was chosen due to its distributed state representation, which mitigate the exponential computational complexity of diagnosers based on observers automata. Moreover, methods for the conversion of Petri net diagnosers into programming languages for implementation on programmable logic controllers (PLC) have been

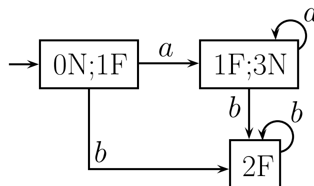


Figure 2.23: Diagnoser automaton G_{d_1} of Example 2.13.

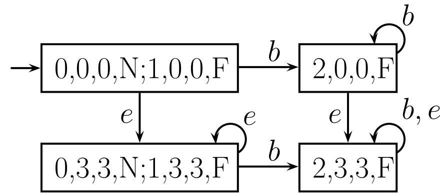


Figure 2.24: Diagnoser automaton G_d of Example 2.13.

proposed in the literature [26].

The problem of failure diagnosis of DESs modeled as automata was also presented in this chapter. We have shown three diagnosis architectures: (*i*) the monolithic diagnosis [14]; (*ii*) the decentralized diagnosis, as defined in Protocol 3 of DEBOUK *et al.* [17]; and (*iii*) the modular diagnosis scheme [59]. Each one of these architectures lead to different notions of diagnosability, namely the monolithic diagnosability, codiagnosability and modular diagnosability.

In the next two chapters, a new architecture for diagnosis of DESs modeled as automata is proposed. Differently from SAMPATH *et al.* [14] and DEBOUK *et al.* [17], this architecture is based on the nonfailure models of the system components instead of the global plant model. The diagnosis scheme proposed in this work is called synchronous diagnosis, and can be implemented in a centralized and decentralized way. Since all system component models are used in the synchronous diagnosis, this approach is also different from the method proposed in CONTANT *et al.* [59], which can be seen as a particular case of the synchronous decentralized diagnosis scheme. The comparison between the notions of modular diagnosability and synchronous codiagnosability is carried out in Chapter 4.

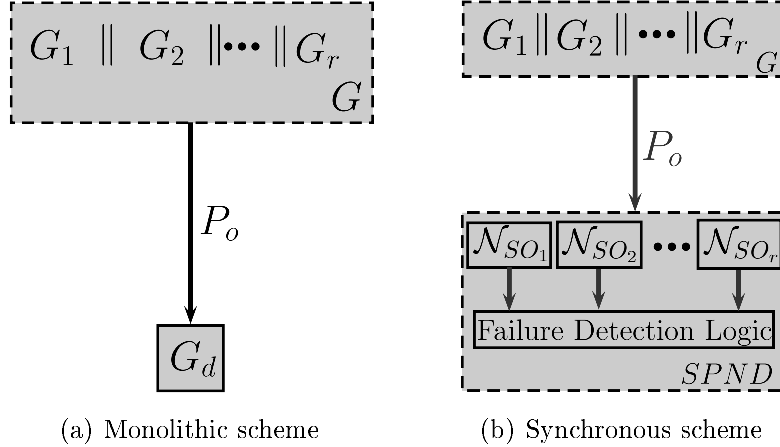
Chapter 3

Synchronous centralized diagnosability of DESs

In CABRAL *et al.* [26], an online diagnoser that provides the state estimate of the nonfailure part of the system, G_N, after the occurrence of an observable trace, is presented. If an observed trace is not feasible in G_N, then the occurrence of the failure event is detected by the online diagnoser, as it is presented in Section 2.4. Although the method presented in CABRAL *et al.* [26] avoids the offline computation of all possible state estimates of the diagnoser, it requires the computation of G_N that may exhibit a large number of states, since it is obtained from the parallel composition of the nonfailure behavior models of the system components.

In order to avoid the computation of the global plant model for diagnosis, in this work, we propose a diagnosis scheme that takes advantage of the modularity of discrete-event systems modeled as automata. In order to do so, we propose a diagnosis method based on the observation of the nonfailure behavior of the system components, modeled by G_{N_k}, for $k = 1, \ldots, r$, where r is the total number of system components [73, 74]. In this regard, let $G = \|_{k=1}^{r} G_k$ be a composed system, *i.e.*, a system that is obtained from the parallel composition of several components, where $G_k = (Q_k, \Sigma_k, f_k, q_{0,k})$, $k = 1, \ldots, r$, denote the automaton models of the system components. Let $\Sigma_k = \Sigma_{k,o} \dot{\cup} \Sigma_{k,uo}$ be the set of events of G_k, where $\Sigma_{k,o}$ and $\Sigma_{k,uo}$ are the set of observable and unobservable events of G_k, respectively.

The diagnoser proposed in this work, called synchronized Petri net diagnoser (SPND), is computed based on G_{N_k}, for $k = 1, \ldots, r$, and provides a superset of the state estimate of the nonfailure behavior model G_N after the occurrence of an observable event. Petri net state observers $\mathcal{N}_{SO_k}$, for $k = 1, \ldots, r$, that estimate online the state of G_{N_k} are constructed, and the occurrence of the failure event is indicated by using a failure detection logic that detects the failure event occurrence when, in at least one $\mathcal{N}_{SO_k}$ for $k \in \{1, \ldots, r\}$, the state estimate is equal to the empty set, *i.e.*, when an observable event $\sigma_o \in \Sigma_{k,o}$ that is not possible in the current state



(a) Monolithic scheme (b) Synchronous scheme

Figure 3.1: Comparison between the monolithic diagnosis architecture (a); and the synchronous diagnosis architecture (b).

estimate of G_{N_k} is executed. Since the state estimate of the SPND is a superset of the actual state estimate of G_N, if an event that is not feasible in the current SPND state estimate is observed, the failure event certainly has occurred.

In Figure 3.1 we show the monolithic and the synchronous diagnosis architectures. In the synchronous diagnosis scheme, all information of the occurrence of observable events is sent to the diagnoser by a unique communication channel, which implies that an observable event $\sigma_o \in \Sigma_o$ is observable for all components for which σ_o is defined, *i.e.*, $\Sigma_{i,o} \cap \Sigma_j \subseteq \Sigma_{j,o}$, for any $i, j \in \{1, 2, \ldots, r\}$. The Petri net state observers $\mathcal{N}_{SO_k}$, for $k = 1, \ldots, r$, are naturally synchronized online by the observable events executed by the system.

Notice that, since the SPND provides a superset of the state estimate of G_N, then it is possible for L to be monolithically diagnosable and not diagnosable by using the synchronous diagnosis scheme presented in Figure 3.1(b). Thus, it is necessary to introduce the definition of synchronous diagnosability of the language of the system. This definition is presented in the next section.

3.1 Synchronous diagnosability

In order to present the definition of synchronous diagnosability of a DES it is necessary first to state the following lemma that shows that $\mathcal{L}(Obs(\|_{k=1}^{r} G_k, \Sigma_o)) \subseteq \mathcal{L}(\|_{k=1}^{r} Obs(G_k, \Sigma_{k,o}))$. This property has been used in the context of supervisory control and it is presented in [75], and appears as an exercise, without a proof, in [76].

Lemma 3.1 *Let $G = \|_{k=1}^{r} G_k$, where $G_k = (Q_k, \Sigma_k, f_k, q_{0,k})$, for $k = 1, 2, \ldots, r$, and let $\Sigma_{k,o} \subseteq \Sigma_k$ denote the set of observable events of G_k. Let $Obs(G_k, \Sigma_{k,o})$ denote*

the observer of G_k. Then,

$$\mathcal{L}(Obs(\|_{k=1}^r G_k, \Sigma_o)) \subseteq \mathcal{L}(\|_{k=1}^r Obs(G_k, \Sigma_{k,o})), \tag{3.1}$$

where $\Sigma_o = \cup_{k=1}^r \Sigma_{k,o}$.

Proof. Let L and L_k be the languages generated by G and G_k, respectively, for $k = 1, \ldots, r$, and consider the projections $P_o : \Sigma^\star \to \Sigma_o^\star$, $P_{k,o}^k : \Sigma_k^\star \to \Sigma_{k,o}^\star$, $P_k : \Sigma^\star \to \Sigma_k^\star$, and $P_{k,o}^o : \Sigma_o^\star \to \Sigma_{k,o}^\star$, for $k = 1, \ldots, r$, where $\Sigma = \cup_{k=1}^r \Sigma_k$, and $\Sigma_o = \cup_{k=1}^r \Sigma_{k,o}$.

Since $G = \|_{k=1}^r G_k$, then, by the definition of parallel composition, $\mathcal{L}(G) = \mathcal{L}(\|_{k=1}^r G_k) = \cap_{k=1}^r P_k^{-1}(L_k)$. Thus,

$$\mathcal{L}(Obs(\|_{k=1}^r G_k, \Sigma_o)) = P_o\left(\cap_{k=1}^r P_k^{-1}(L_k)\right). \tag{3.2}$$

Let us now consider $\mathcal{L}\left(\|_{k=1}^r Obs(G_k, \Sigma_{k,o})\right)$. Since $\mathcal{L}(Obs(G_k, \Sigma_{k,o})) = P_{k,o}^k(L_k)$, then $\mathcal{L}\left(\|_{k=1}^r Obs(G_k, \Sigma_{k,o})\right) = \cap_{k=1}^r P_{k,o}^{o^{-1}}\left(P_{k,o}^k(L_k)\right)$.

According to the definitions of $P_{k,o}^o$, P_k, and P_o, it can be seen that

$$P_{k,o}^{o^{-1}}(s) = P_o(P_k^{-1}(s)),$$

for all $s \in \Sigma_{k,o}^\star$. Thus,

$$P_{k,o}^{o^{-1}}(P_{k,o}^k(L_k)) = P_o(P_k^{-1}(P_{k,o}^k(L_k))), \tag{3.3}$$

for $k = 1, 2, \ldots, r$.

Notice that, since $L_k \subseteq \Sigma_k^\star$, $\Sigma_k \subseteq \Sigma$, and $\Sigma_{k,o} \subseteq \Sigma_o$, $P_{k,o}^k(L_k) = P_o(L_k)$, and

$$P_o(P_k^{-1}(P_{k,o}^k(L_k))) = P_o(P_k^{-1}(P_o(L_k))), \tag{3.4}$$

for $k = 1, 2, \ldots, r$. In addition, notice that

$$P_o(P_k^{-1}(P_o(s))) = P_o(P_k^{-1}(s)), \tag{3.5}$$

for all $s \in L_k$. Thus,

$$P_o(P_k^{-1}(P_o(L_k))) = P_o(P_k^{-1}(L_k)), \tag{3.6}$$

for $k = 1, 2, \ldots, r$.

According to Eq. (3.3),

$$\cap_{k=1}^r P_{k,o}^{o^{-1}}\left(P_{k,o}^k(L_k)\right) = \cap_{k=1}^r P_o(P_k^{-1}(P_{k,o}^k(L_k))). \tag{3.7}$$

Notice that the right-hand side of Eq. (3.7) can be rewritten according to Eq.

(3.4), as follows:

$$\cap_{k=1}^{r} P_o(P_k^{-1}(P_{k,o}^{k}(L_k))) = \cap_{k=1}^{r} P_o(P_k^{-1}(P_o(L_k))). \tag{3.8}$$

Now, using Eq. (3.6) in Eq. (3.8) we have that:

$$\cap_{k=1}^{r} P_o(P_k^{-1}(P_o(L_k))) = \cap_{k=1}^{r} P_o(P_k^{-1}(L_k)), \tag{3.9}$$

and, therefore,

$$\mathcal{L}(\|_{k=1}^{r} Obs(G_k, \Sigma_{k,o})) = \cap_{k=1}^{r} P_{k,o}^{o^{-1}}\left(P_{k,o}^{k}(L_k)\right) = \cap_{k=1}^{r} P_o(P_k^{-1}(L_k)). \tag{3.10}$$

Finally, from property (2.1), it can be seen that:

$$P_o\left(\cap_{k=1}^{r} P_k^{-1}(L_k)\right) \subseteq \cap_{k=1}^{r} P_o(P_k^{-1}(L_k)).$$

Therefore, according to Eqs. (3.2) and (3.10),

$$\mathcal{L}(Obs(\|_{k=1}^{r} G_k, \Sigma_o)) \subseteq \mathcal{L}(\|_{k=1}^{r} Obs(G_k, \Sigma_{k,o})). \tag{3.11}$$

■

According to Lemma 3.1, the language generated by $Obs(\|_{k=1}^{r} G_{N_k}, \Sigma_o)$ is a subset of the language generated by the parallel composition of $Obs(G_{N_k}, \Sigma_{k,o})$, for $k = 1, \ldots, r$. In the following corollary we present a condition that ensures the equality in (3.1).

Corollary 3.1 *Let $\Sigma_{k,uo}$ be the set of unobservable events of Σ_k, for $k = 1, \ldots, r$. If $\Sigma_{i,uo} \cap \Sigma_{j,uo} = \emptyset$, for all $i \neq j$ and $i, j \in \{1, \ldots, r\}$, then $\mathcal{L}(Obs(\|_{k=1}^{r} G_k, \Sigma_o)) = \mathcal{L}(\|_{k=1}^{r}(Obs(G_k, \Sigma_{k,o})))$.*

Proof. In order to prove that $\mathcal{L}(Obs(\|_{k=1}^{r} G_k, \Sigma_o)) = \mathcal{L}(\|_{k=1}^{r} Obs(G_k, \Sigma_{k,o}))$, it suffices to prove that $\mathcal{L}(Obs(\|_{k=1}^{r} G_k, \Sigma_o)) \supseteq \mathcal{L}(\|_{k=1}^{r} Obs(G_k, \Sigma_{k,o}))$, when $\Sigma_{i,uo} \cap \Sigma_{j,uo} = \emptyset$ for all $i \neq j$ and $i, j \in \{1, \ldots, r\}$, since in Lemma 3.1 we have already proved that $\mathcal{L}(Obs(\|_{k=1}^{r} G_k, \Sigma_o)) \subseteq \mathcal{L}(\|_{k=1}^{r} Obs(G_k, \Sigma_{k,o}))$. In order to do so, let $\eta \in \mathcal{L}(\|_{k=1}^{r} Obs(G_k, \Sigma_{k,o}))$, which, in accordance with Eq. (3.10), implies that $\eta \in \cap_{k=1}^{r} P_o(P_k^{-1}(L_k))$, *i.e.*, there exist traces $s_k \in P_k^{-1}(L_k)$, for $k = 1, \ldots, r$ such that $P_o(s_1) = P_o(s_2) = \ldots = P_o(s_r) = \eta$. Let $K_k \subseteq P_k^{-1}(L_k)$ be languages such that $\eta = P_o(s_k)$, for any $k \in \{1, \ldots, r\}$. Since $\Sigma_{i,uo} \cap \Sigma_{j,uo} = \emptyset$, for all $i \neq j$ and $i, j \in \{1, \ldots, r\}$, then all unobservable events of the set $\Sigma_{k,uo}$ are private events of Σ_k, for $k = 1, \ldots, r$, which implies that $\cap_{k=1}^{r} K_k \neq \emptyset$. Thus, $\eta \in \cap_{k=1}^{r} P_o(P_k^{-1}(L_k)$, which, in accordance with Eq. (3.2), implies that $\eta \in \mathcal{L}(Obs(\|_{k=1}^{r} G_k, \Sigma_o))$. ■

It is important to remark that the condition presented in Corollary 3.1 is only sufficient, *i.e.*, even if $\Sigma_{i,uo} \cap \Sigma_{j,uo} \neq \emptyset$ for any $i \neq j$ and $i, j \in \{1, \ldots, r\}$, the language $\mathcal{L}(Obs(\|_{k=1}^{r} G_k, \Sigma_o))$ can be equal to $\mathcal{L}(\|_{k=1}^{r} Obs(G_k, \Sigma_{k,o}))$.

In this work, the same strategy for diagnosis proposed in CABRAL *et al.* [26] is used, namely, the diagnoser provides the current state estimate of the nonfailure behavior of the system, and, when an event that is not feasible in the current state estimate is observed, the diagnoser indicates the occurrence of the failure event. However, differently from CABRAL *et al.* [26], we exploit the composed structure of the system, *i.e.*, the online state estimate of each module is carried out by the diagnoser that naturally synchronizes the state estimate of the modules based on the occurrence of observable events, whose language is $\mathcal{L}(\|_{k=1}^{r} Obs(G_{N_k}, \Sigma_{k,o})) = \cap_{k=1}^{r} P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))$, where $P_{k,o}^{o} : \Sigma_o^\star \rightarrow \Sigma_{k,o}^\star$, $P_{k,o} : \Sigma^\star \rightarrow \Sigma_{k,o}^\star$, and $\Sigma_o = \cup_{k=1}^{r} \Sigma_{k,o}$.

Let L_{N_a} denote the augmented nonfailure language obtained by using the synchronous diagnosis approach, *i.e.*, $L_{N_a} = \cap_{k=1}^{r} P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))$. According to Equation (3.1),

$$P_o(L_N) \subseteq L_{N_a}, \tag{3.12}$$

which shows that a diagnoser that uses the information provided by the parallel composition of the observers of the system modules may represent more observable traces than the system is capable to generate. Indeed, the diagnosis based on the observation of the system modules is equivalent to the diagnosis of an augmented system G_a whose generated language is $L_a = L_{N_a} \cup \overline{L}_F$, where L_F is the failure language of the system. Therefore, if there exists a nonfailure trace in $L_{N_a} \setminus L_N$ with the same projection as an arbitrarily long length failure trace in L_F, then L_a is not diagnosable even if L is diagnosable. This leads to the following definition of synchronous diagnosability of DESs.

Definition 3.1 (Synchronous diagnosability) *Let L and $L_N \subset L$ denote the languages generated by G and G_N, respectively, and let $L_F = L \setminus L_N$. Consider that the system is composed of r modules, such that $G_N = \|_{k=1}^{r} G_{N_k}$, where G_{N_k} is the automaton that models the nonfailure behavior of G_k, and let L_{N_k} denote the language generated by G_{N_k}, for $k = 1, \ldots, r$. Then, L is said to be synchronously diagnosable with respect to L_{N_k}, $P_{k,o}^{o} : \Sigma_o^\star \rightarrow \Sigma_{k,o}^\star$, $P_{k,o} : \Sigma^\star \rightarrow \Sigma_{k,o}^\star$, for $k = 1, \ldots, r$, $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$, and Σ_f if*

$$(\exists z \in \mathbb{N})(\forall s \in L_F)(\forall st \in L_F, \|t\| \geq z) \Rightarrow$$
$$(P_o(st) \notin \cap_{k=1}^{r} P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))).$$

Notice that Definition 3.1 of synchronous diagnosability of a language L is equiv-

alent to the standard definition of diagnosability (Definition 2.22) of a language $L_a = \overline{L}_F \cup L_{N_a}$, where $L_{N_a} = \cap_{k=1}^{r} P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))$. This leads to the following theorem that establishes the relation between the notions of diagnosability (Definition 2.22) and synchronous diagnosability (Definition 3.1).

Theorem 3.1 *If L is synchronous diagnosable with respect to L_{N_k}, $P_{k,o}^{o} : \Sigma_o^\star \to \Sigma_{k,o}^\star$, $P_{k,o} : \Sigma^\star \to \Sigma_{k,o}^\star$, for $k = 1, \ldots, r$, $P_o : \Sigma^\star \to \Sigma_o^\star$, and Σ_f, then L is diagnosable with respect to $P_o : \Sigma^\star \to \Sigma_o^\star$ and Σ_f, where $\Sigma_o = \cup_{k=1}^{r} \Sigma_{k,o}$.*

Proof. According to Definition 3.1, in order to L be synchronously diagnosable, there cannot be an arbitrarily long failure trace st such that $P_o(st) \in L_{N_a}$ and, according to Definition 2.22, in order to L be diagnosable, there cannot be an arbitrarily long failure trace st such that $P_o(st) \in P_o(L_N)$. Since, according to Equation (3.12), $P_o(L_N) \subseteq L_{N_a}$, if $P_o(st) \notin L_{N_a}$ then $P_o(st) \notin P_o(L_N)$, which implies that if L is synchronously diagnosable, L is diagnosable. ■

In the next section we present an algorithm for the verification of synchronous diagnosability of the language of a composed system.

3.2 Synchronous diagnosability verifier

In order to present the algorithm for the verification of the synchronous diagnosability, we first show an algorithm for the computation of the nonfailure behavior models G_{N_k} from the system components models G_k, and the system model G.

Algorithm 3.1 *Nonfailure behavior models of the system components.*

Input: $G_k = (Q_k, \Sigma_k, f_k, q_{0,k})$, *for* $k = 1, \ldots, r$, *and* $G = (Q, \Sigma, f, q_0)$.
Output: $G_{N_k} = (Q_{N_k}, \Sigma_{N_k}, f_{N_k}, q_{0,N_k})$, *for* $k = 1, \ldots, r$.

1: *Compute automaton $G_N = (Q_N, \Sigma_N, f_N, q_0)$ according to Algorithm 2.4 [71].*

2: *For all transitions $f_N(q_N, \sigma) = q'_N$ in G_N, flag the transitions $f_k(q_k, \sigma) = q'_k$ in G_k, for $k = 1, \ldots, r$, where q_k and q'_k are the k-th elements of q_N and q'_N, respectively.*

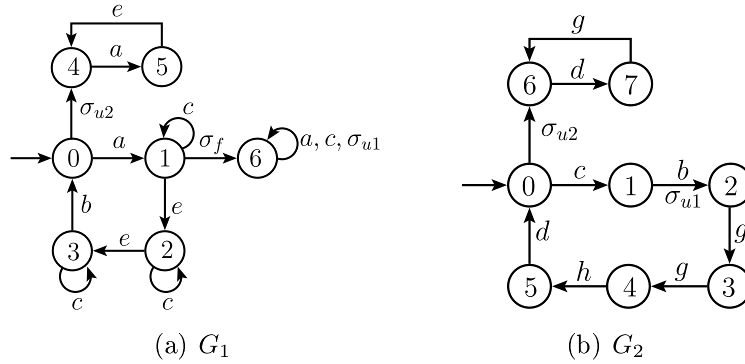
3: *Obtain automata G'_k by erasing from G_k all transitions that are not flagged.*

4: *Compute automata $G_{N_k} = Ac(G'_k) = (Q_{N_k}, \Sigma_{N_k}, f_{N_k}, q_{0,N_k})$, for $k = 1, \ldots, r$.*

5: *Redefine the event sets $\Sigma_{N_k} \leftarrow \Sigma_k \setminus \Sigma_f$, for $k = 1, \ldots, r$.*

According to Algorithm 3.1, the nonfailure behavior models of the system components G_{N_k} are obtained from the composed system $G_N = \|_{k=1}^{r} G_{N_k}$. The construction

of G_{N_k} from G_N is necessary since the post-failure behavior of a failure component G_i can interact with another component G_j, $i \neq j$, where the failure event is not modeled. If, due to the interaction between modules G_i and G_j, the behavior of G_j after the failure event occurrence is different from its behavior when the failure event does not occur, G_j can be different from G_{N_j}, even if the failure event is not modeled in G_j. The following example illustrates this problem.



(a) G_1 (b) G_2

Figure 3.2: Automata G_1 and G_2 of Example 3.1.

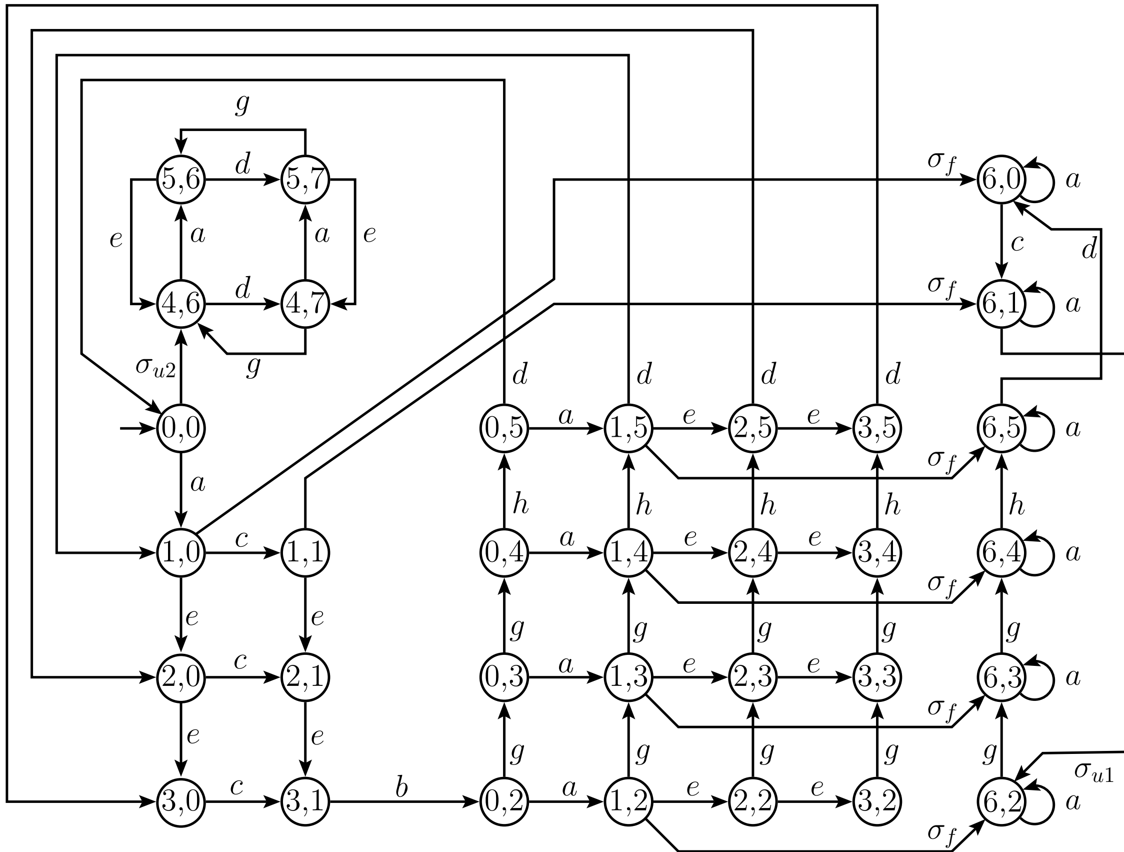


Figure 3.3: Automaton G of Example 3.1.

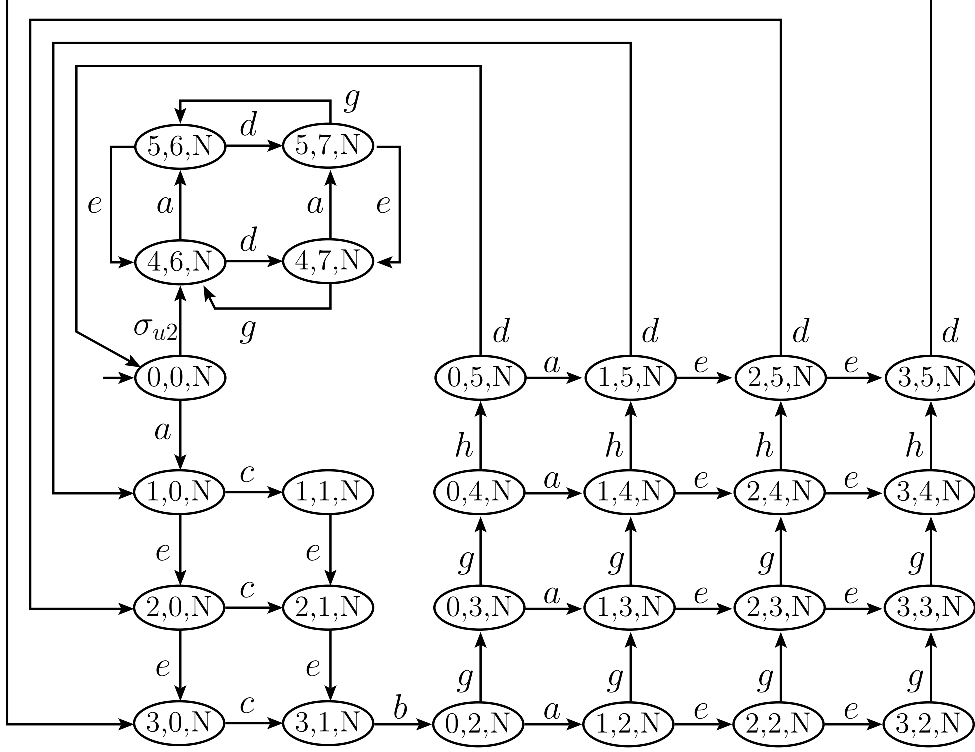


Figure 3.4: Automaton G_N of Example 3.1.

Example 3.1 *Consider the system $G = G_1 \| G_2$, where G_1 and G_2 are depicted in Figures 3.2(a) and 3.2(b), respectively, and automaton G is shown in Figure 3.3. The event sets of G, G_1 and G_2 are $\Sigma = \{a, b, c, d, e, g, h, \sigma_{u1}, \sigma_{u2}, \sigma_f\}$, $\Sigma_1 = \Sigma_{1,o} \cup \Sigma_{1,uo} = \{a, b, c, e, \sigma_{u1}, \sigma_{u2}, \sigma_f\}$, and $\Sigma_2 = \Sigma_{2,o} \cup \Sigma_{2,uo} = \{b, d, g, h, \sigma_{u1}, \sigma_{u2}\}$, respectively, where $\Sigma_{1,o} = \{a, b, c, e\}$, $\Sigma_{1,uo} = \{\sigma_{u1}, \sigma_{u2}, \sigma_f\}$, $\Sigma_{2,o} = \{b, d, g, h\}$, and $\Sigma_{1,uo} = \{\sigma_{u1}, \sigma_{u2}\}$. The set of failure events $\Sigma_f = \{\sigma_f\}$. Automaton G_N, that models the nonfailure behavior of G obtained by following Step 1 of Algorithm 3.1, is presented in Figure 3.4. Notice that transition $(1, \sigma_{u1}, 2)$ of automaton G_2 can only occur after the occurrence of the failure event σ_f. Moreover, σ_f belongs only to the event set of automaton G_1, and thus, although the failure event is not modeled in G_2, the transition $(1, \sigma_{u1}, 2)$ of G_2 does not belong to its nonfailure behavior. In Figure 3.5, we show automata G_{N_1} and G_{N_2} obtained by following Step 4 of Algorithm 3.1.*

The following algorithm can be used to verify the synchronous diagnosability of the language of a DES.

Algorithm 3.2 *Synchronous Diagnosability Verification*

Input: *System modules G_k, for $k = 1, \ldots, r$, and $G = \|_{k=1}^{r} G_k$.*
Output: *Synchronous diagnosability decision.*

1: Compute automaton G_F that models the failure behavior of G, whose marked language is $L_F = L \setminus L_N$, according to Algorithm 2.7 [71].

2: Compute automata G_{N_k} by following the steps of Algorithm 3.1.

3: Compute automaton $G_N^R = (Q_N^R, \Sigma^R, f_N^R, q_0)$ as follows:

3.1: Define function $R_k : \Sigma_{N_k} \to \Sigma_{N_k}^R$, as:

$$R_k(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_{k,o}, \\ \sigma_{R_k}, & \text{if } \sigma \in \Sigma_{k,uo}. \end{cases} \tag{3.13}$$

3.2: Construct automata $G_{N_k}^R = (Q_{N_k}, \Sigma_{N_k}^R, f_{N_k}^R, q_{0,N_k})$, $k = 1, \ldots, r$, with $f_{N_k}^R(q_{N_k}, R_k(\sigma)) = f_{N_k}(q_{N_k}, \sigma)$, $\forall q_{N_k} \in Q_{N_k}$ and $\forall \sigma \in \Sigma_{N_k}$.

3.3: Compute $G_N^R = \|_{k=1}^r G_{N_k}^R$.

4: Compute the verifier automaton $G_V^{SD} = (Q_V, \Sigma_V, f_V, q_{0,V}) = G_F \| G_N^R$. Notice that a state of G_V^{SD} is given by $q_V = (q_F, q_N^R)$, where q_F and q_N^R are states of G_F and G_N^R, respectively, and $q_F = (q, q_l)$, where $q \in Q$ and $q_l \in \{N, F\}$.

5: Verify the existence of a cyclic path $cl = (q_V^\delta, \sigma_\delta, q_V^{\delta+1}, \ldots, q_V^\gamma, \sigma_\gamma, q_V^\delta)$, where $\gamma \geq \delta > 0$, in G_V^{SD} such that:

$$\exists j \in \{\delta, \delta+1, \ldots, \gamma\} \text{ such that for some } q_V^j,$$
$$(q_l^j = F) \wedge (\sigma_j \in \Sigma).$$

If the answer is yes, then L is not synchronously diagnosable with respect to L_{N_k}, $P_{k,o}^o : \Sigma_o^\star \to \Sigma_{k,o}^\star$, $P_{k,o} : \Sigma^\star \to \Sigma_{k,o}^\star$, for $k = 1, \ldots, r$, $P_o : \Sigma^\star \to \Sigma_o^\star$, and Σ_f. Otherwise, L is synchronously diagnosable.

The method to verify the synchronous diagnosability of a system is based on the comparison between automata G_F and G_N^R. Automaton G_F models the failure behavior of the system G and automaton G_N^R models the augmented nonfailure

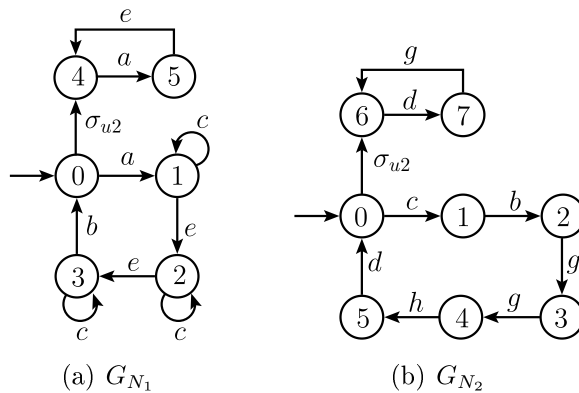


Figure 3.5: Automata G_{N_1} and G_{N_2} of Example 3.1.

behavior considered in the synchronous diagnosis scheme, such that the projection in Σ_o of the language generated by G_N^R, $P_o^R(\mathcal{L}(G_N^R))$, where $P_o^R : \Sigma^{R\star} \rightarrow \Sigma_o^\star$, is equal to the nonfailure language observed by the synchronous diagnoser, *i.e.*, $P_o^R(\mathcal{L}(G_N^R)) = L_{N_a} = \cap_{k=1}^r P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))$.

In order to prove the correctness of Algorithm 3.2 we first present the following lemmas.

Lemma 3.2 *Let G_{N_k} be the automaton that models the nonfailure behavior of G_k, and L_{N_k} be the language generated by G_{N_k}, for $k = 1, \ldots, r$. Let $G_{N_k}^R$ be the automaton obtained from G_{N_k} by applying Step 3 of Algorithm 3.2, and $L_{N_k}^R$ be the language generated by $G_{N_k}^R$, for $k = 1, \ldots, r$. Then, $P_o^R[\cap_{k=1}^r P_{N_k}^{R^{-1}}(L_{N_k}^R)] = \cap_{k=1}^r P_o^R[P_{N_k}^{R^{-1}}(L_{N_k}^R)]$, where $P_o^R : \Sigma^{R\star} \rightarrow \Sigma_o^\star$, $P_{N_k}^R : \Sigma^{R\star} \rightarrow \Sigma_{N_k}^\star$, for $k = 1, \ldots, r$, and $\Sigma^R = \cup_{k=1}^r \Sigma_{N_k}^R$.*

Proof. In order to show that $P_o^R[\cap_{k=1}^r P_{N_k}^{R^{-1}}(L_{N_k}^R)] = \cap_{k=1}^r P_o^R[P_{N_k}^{R^{-1}}(L_{N_k}^R)]$, we must only proof that $P_o^R[\cap_{k=1}^r P_{N_k}^{R^{-1}}(L_{N_k}^R)] \supseteq \cap_{k=1}^r P_o^R[P_{N_k}^{R^{-1}}(L_{N_k}^R)]$, since $P_o^R[\cap_{k=1}^r P_{N_k}^{R^{-1}}(L_{N_k}^R)] \subseteq \cap_{k=1}^r P_o^R[P_{N_k}^{R^{-1}}(L_{N_k}^R)]$ is true, according to Equation (2.1).

Let $s = \sigma_{o_1}\sigma_{o_2}\ldots\sigma_{o_n}$ be a trace of events such that $s \in \cap_{k=1}^r P_o^R[P_{N_k}^{R^{-1}}(L_{N_k}^R)]$. Thus, $s \in P_o^R[P_{N_k}^{R^{-1}}(L_{N_k}^R)]$ for all $k = 1, \ldots, r$. Therefore, there exists at least one trace $y_k \in P_{N_k}^{R^{-1}}(L_{N_k}^R)$, for $k \in \{1, \ldots, r\}$, such that $P_o(y_1) = P_o(y_2) = \ldots = P_o(y_r) = s$. Since $P_o(y_1) = P_o(y_2) = \ldots = P_o(y_r) = s$, y_k can be written as $y_k = v_1\sigma_{o_1}v_2\sigma_{o_2}v_3\ldots v_n\sigma_{o_n}v_{n+1}$, where $v_i \in (\Sigma^R \setminus \Sigma_o)^\star$. Moreover, since $y_k \in P_{N_k}^{R^{-1}}(L_{N_k}^R)$, v_i can be written as $v_i = t_1\sigma_{k,1}^R t_2\sigma_{k,2}^R\ldots t_\eta$, where $\sigma_{k,j}^R \in \Sigma_{N_k}^R$, $t_i \in (\Sigma_N^R \setminus \Sigma_{N_1}^R)^\star$. Therefore, there exists at least one trace $y \in \cap_{k=1}^r P_{N_k}^{R^{-1}}(L_{N_k}^R)$ with the same observation as y_k for $k = 1, \ldots, r$. Since $P_o^R(y) = s$, then $s \in P_o^R[\cap_{k=1}^r P_{N_k}^{R^{-1}}(L_{N_k}^R)]$, which concludes the proof. ■

Lemma 3.3 *Let L_{N_k} be the language generated by G_{N_k}. Then,*

$$\cap_{k=1}^r P_o^R[P_{N_k}^{R^{-1}}(L_{N_k}^R)] = \cap_{k=1}^r P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k})).$$

Proof. Notice that $L_{N_k}^R$ is obtained from L_{N_k} by renaming its unobservable events. Therefore, since $P_{k,o}^o : \Sigma_o^\star \rightarrow \Sigma_{k,o}^\star$, $P_{k,o} : \Sigma^\star \rightarrow \Sigma_{k,o}^\star$, $P_{N_k}^R : \Sigma^{R\star} \rightarrow \Sigma_{N_k}^\star$, and $P_o^R : \Sigma^{R\star} \rightarrow \Sigma_o^\star$, it is straightforward to see that $P_o^R[P_{N_k}^{R^{-1}}(L_{N_k}^R)] = P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))$, for $k = 1, \ldots, r$. ■

We can now state the following theorem that proves the correctness of Algorithm 3.2.

Theorem 3.2 *Let L_{N_k} denote the language generated by G_{N_k}, for $k = 1, \ldots, r$. Then, L is not synchronously diagnosable with respect to L_{N_k}, $P_{k,o}^o : \Sigma_o^\star \rightarrow \Sigma_{k,o}^\star$,*

$P_{k,o}: \Sigma^\star \rightarrow \Sigma_{k,o}^\star$, for $k = 1, \ldots, r$, $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$, and Σ_f if, and only if, there exists a cyclic path $cl = (q_V^\delta, \sigma_\delta, q_V^{\delta+1}, \ldots, q_V^\gamma, \sigma_\gamma, q_V^\delta)$ in G_V^{SD}, where $\gamma \geq \delta > 0$, such that:

$$\exists j \in \{\delta, \delta+1, \ldots, \gamma\} \ s.t.\ for\ some\ q_V^j,$$
$$(q_l^j = F) \wedge (\sigma_j \in \Sigma). \quad (3.14)$$

Proof. According to Definition 3.1, in order to verify the synchronous diagnosability of the language of the system L, it is necessary to check if there exists an arbitrarily long failure trace st such that $P_o(st) \in \cap_{k=1}^r P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))$. Lemma 3.2 shows that $P_o^R[\cap_{k=1}^r P_{N_k}^{R^{-1}}(L_{N_k}^R)] = \cap_{k=1}^r P_o^R[P_{N_k}^{R^{-1}}(L_{N_k}^R)]$, and according to Lemma 3.3, $\cap_{k=1}^r P_o^R[P_{N_k}^{R^{-1}}(L_{N_k}^R)] = \cap_{k=1}^r P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))$. Therefore, $\cap_{k=1}^r P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k})) = P_o^R[\cap_{k=1}^r P_{N_k}^{R^{-1}}(L_{N_k}^R)]$. Thus, in order to check the synchronous diagnosability of L, it can be verified if there exists a failure trace st such that $P_o(st) \in P_o^R[\cap_{k=1}^r P_{N_k}^{R^{-1}}(L_{N_k}^R)]$. Since the unobservable events of G_N^R are renamed, and hence, are private events of G_N^R, it can be seen that the verifier automaton G_V^{SD} proposed here is equal to the verifier automaton G_V obtained by applying the method proposed in MOREIRA *et al.* [71] to a system whose failure automaton marks L_F and whose observable nonfailure behavior automaton generates L_{N_a}. Moreover, the same necessary and sufficient condition (3.14) would be obtained by using the verification method proposed in MOREIRA *et al.* [71], which concludes the proof. ■

Notice that the construction of G_V^{SD} according to Algorithm 3.2 is polynomial in the number of states of G_{N_k}, and it is exponential in the number of system components. In order to see this fact, let us compute the complexity of each step of Algorithm 3.2.

In Step 1 of Algorithm 3.2, automaton G_F is computed. The number of transitions and states of G_F are bounded by $2 \times |Q|$ and $2 \times |Q| \times |\Sigma|$, respectively [71]. In Step 2 automata G_{N_k}, for $k = 1, \ldots, r$, are computed. Since automata G_{N_k} are obtained from G_N by erasing all transitions that are not associated with a transition of G_k, and taking the accessible part of the result, the number of transitions and states of automata G_{N_k} are bounded by the number of transitions and states of automata G_k. Automata $G_{N_k}^R$, computed in Step 3, has the same number of transitions and states of automata G_{N_k}. Finally, automaton G_V^{SD} is computed by making the parallel composition between automata $G_{N_k}^R$ and G_F. Therefore, the number of states and transitions of automaton G_V^{SD} are, in the worst-case, equal to $(\prod_{k=1}^r |Q_k|) \times |Q|$, and $(\prod_{k=1}^r |Q_k|) \times |Q| \times |\Sigma_V|$, respectively, where $\Sigma_V = \Sigma^R \cup \Sigma$. Thus, the computational complexity of Algorithm 3.2 is $O(r \times (\prod_{k=1}^r |Q_k|) \times |Q| \times |\Sigma|)$.

Although the computational complexity of the synchronous diagnosability verifier G_V^{SD} is exponential in the number of the system components, the main goal in this work is to provide a diagnosis method that is polynomial in the number of com-

ponents of the system, avoiding the exponential growth of the size of the diagnoser for implementation. In the following example, we illustrate the use of Algorithm 3.2 for the verification of synchronous diagnosability.

Example 3.2 *Let G_1 and G_2, whose transition diagrams are shown in Figure 3.2, be the components of a system $G = G_1 \| G_2$, depicted in Figure 3.3. Let $\Sigma = \{a, b, c, d, e, g, h, \sigma_{u1}, \sigma_{u2}, \sigma_f\}$, $\Sigma_o = \{a, b, c, d, e, g, h\}$, $\Sigma_{uo} = \{\sigma_{u1}, \sigma_{u2}, \sigma_f\}$, $\Sigma_f = \{\sigma_f\}$, $\Sigma_1 = \{a, b, c, e, \sigma_{u1}, \sigma_{u2}, \sigma_f\}$, $\Sigma_2 = \{b, c, d, g, h, \sigma_{u1}, \sigma_{u2}, \sigma_f\}$, $\Sigma_{1,o} = \{a, b, c, e\}$, and $\Sigma_{2,o} = \{b, c, d, g, h\}$. In the first step of Algorithm 3.2 automaton G_F, shown in Figure 3.6, is computed. In Step 2, automata G_{N_1} and G_{N_2}, depicted in Figure 3.5, are computed, and, in Step 3, automaton G_N^R, obtained by making the parallel composition of $G_{N_1}^R$ and $G_{N_2}^R$, depicted in Figures 3.7(a) and 3.7(b), respectively, is computed. Notice that the gray states of G_N^R, depicted in Figure 3.8, and their corresponding transitions labeled with observable events, do not belong to G_N, which indicates the growth of the nonfailure language for synchronous diagnosis compared to the monolithic diagnosis scheme. Finally, in Step 4 of Algorithm 3.2, the synchronous verifier automaton G_V^{SD}, presented in Figure 3.9, is computed. Notice that there are no cycles in G_V^{SD} that satisfy condition (3.14) of Theorem 3.2. Thus, L is synchronously diagnosable with respect to L_{N_1}, L_{N_2}, $P_{1,o}^o : \Sigma_o^\star \rightarrow \Sigma_{1,o}^\star$, $P_{2,o}^o : \Sigma_o^\star \rightarrow \Sigma_{2,o}^\star$, $P_{1,o} : \Sigma^\star \rightarrow \Sigma_{1,o}^\star$, $P_{2,o} : \Sigma^\star \rightarrow \Sigma_{2,o}^\star$, $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$, and Σ_f.*

It is important to notice that the sum of the number of states of G_{N_1} and G_{N_2} in this example is equal to 14, and the cardinality of the state space of G_N is 27. This shows that the synchronous diagnosis scheme has a lower cost for implementation than a diagnoser computed from the global nonfailure behavior model of the system G_N.

Since the synchronous diagnosis of L is equivalent to the monolithic diagnosis of the augmented language $L_a = L_{N_a} \cup \overline{L}_F$, where $L_{N_a} \supseteq P_o(L_N)$, then, the delay bound for synchronous diagnosis can be larger than the delay bound for the monolithic diagnosis. In the next section, we present a method for the computation of the delay bound for synchronous diagnosis.

3.3 Delay bound for synchronous diagnosis

In Section 3.1, we have shown that the nonfailure language observed by the synchronous diagnoser can be a larger set than the natural projection of the nonfailure language of the system. This fact can add a delay[1] to the synchronous diagnosis compared with the standard monolithic diagnosis case, which can cause a decrease

[1] In this work, the delay bound is considered as the maximum number of events that the system can execute after the occurrence of the failure event σ_f, until σ_f is detected by the diagnoser.

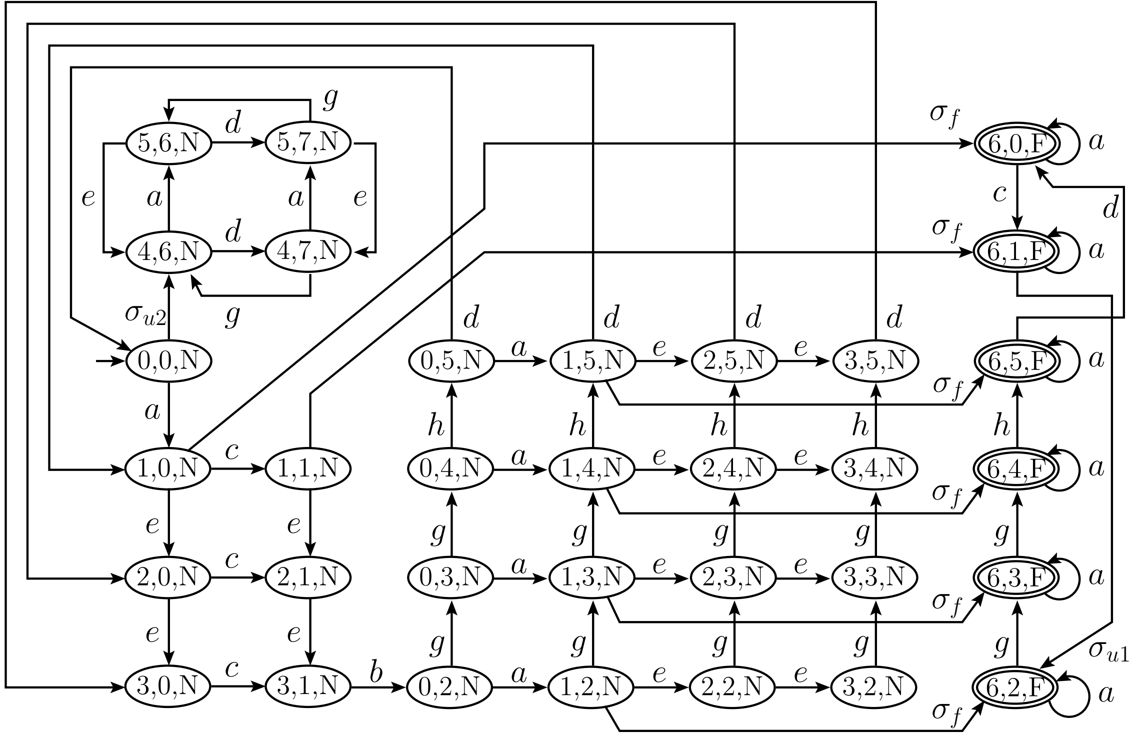


Figure 3.6: Automaton G_F of Example 3.2.

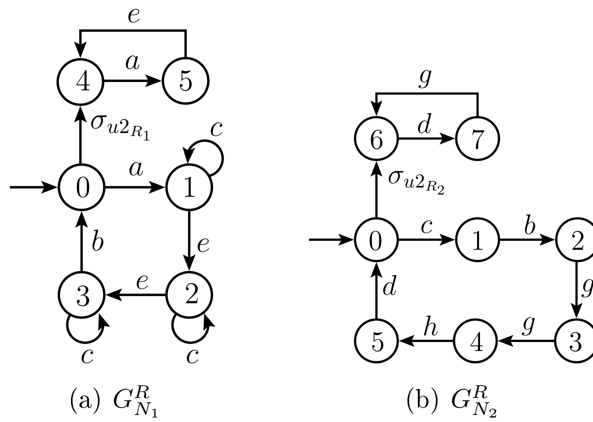


Figure 3.7: Automata $G^R_{N_1}$ and $G^R_{N_2}$ of Example 3.2.

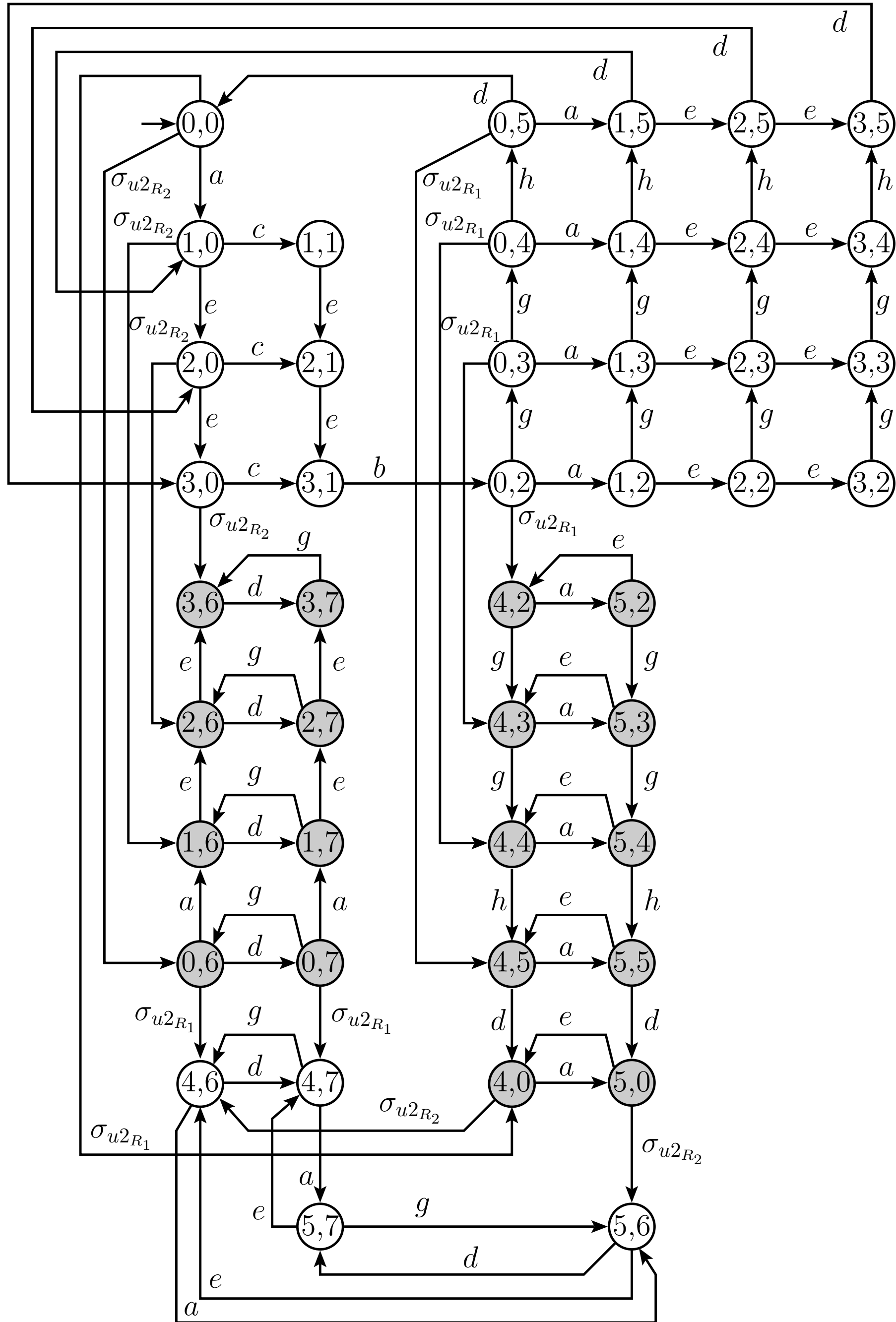


Figure 3.8: Automaton G_N^R of Example 3.2.

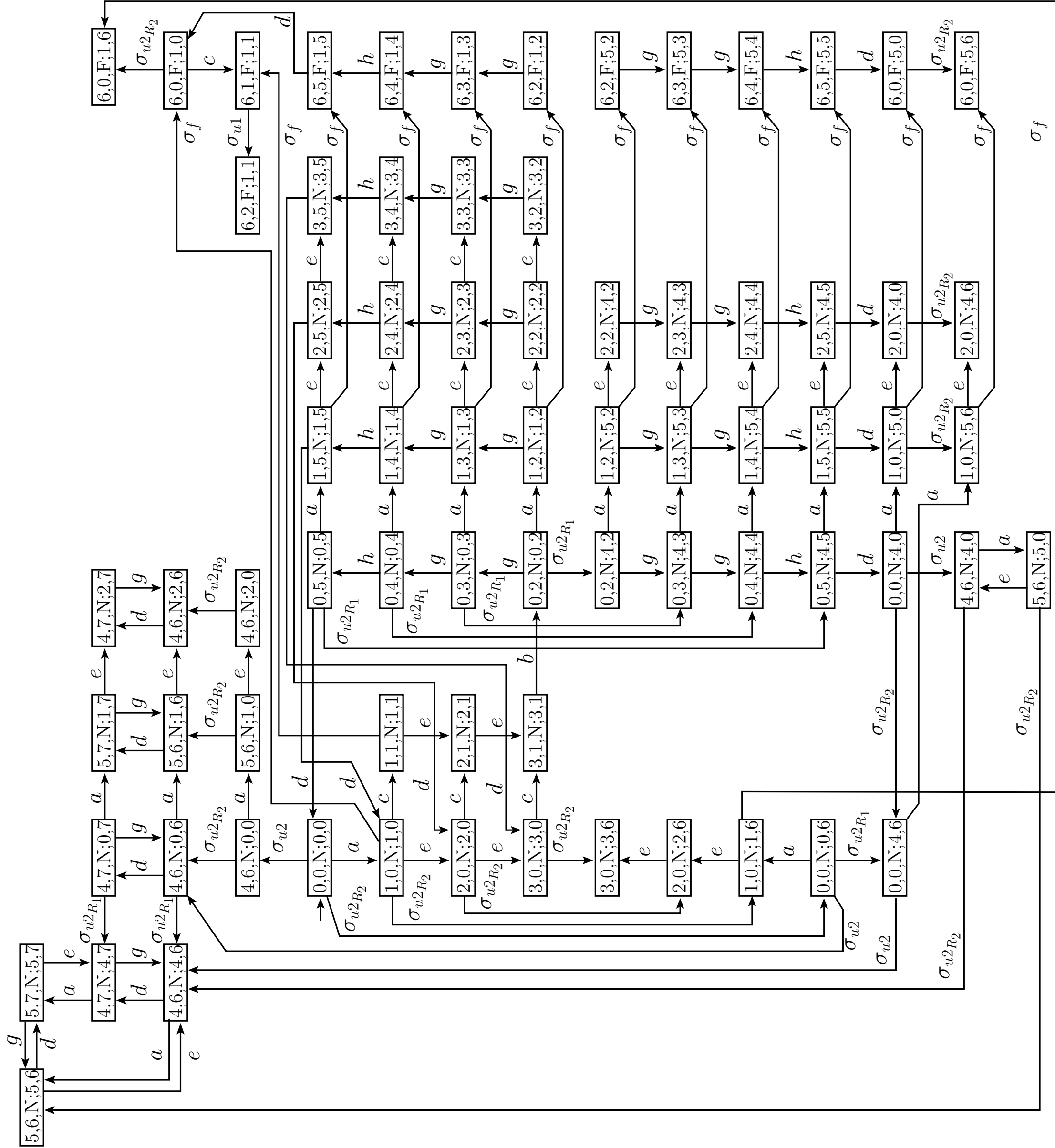


Figure 3.9: Automaton G_V^{SD} of Example 3.2.

in the system performance. Thus, the computation of the delay bound z^* of the synchronous diagnosis is important to evaluate if this method can be implemented in a real system. In this section, we present a polynomial time algorithm for the computation of the delay bound z^* for synchronous diagnosis. The method for the computation of the delay bound presented in this section is based on the method proposed in TOMOLA *et al.* [77] for the computation of the delay bound for robust codiagnosability of DESs.

With a view to computing the delay bound z^*, it is necessary to compute first the maximum number of events d that the system can execute after the occurrence of the failure event σ_f, for which there exists a failure trace st and a nonfailure trace ω, such that $P_o(\omega) \in L_{N_a}$, with the same observation:

$$d = \max\{\|t\| : (s \in L_F)(st \in L_F)(P_o(st) = P_o(\omega), \\ P_o(\omega) \in \cap_{k=1}^{r} P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k})))\}.$$

Remark 3.1 *In order to compute d, it is necessary to find the traces $st \in L_F$ and ω, where $P_o(\omega) \in L_{N_a}$, such that $P_o(st) = P_o(\omega)$, and t has maximum length. Notice that G_V^{SD} represents only the nonfailure traces $P_o(\omega) \in L_{N_a}$ and failure traces st that have the same natural projection P_o [78]. Therefore, the computation of d can be carried out by searching the path of G_V^{SD} associated with a trace in $\Sigma^\star$ with the greatest length after the occurrence of the failure event σ_f.*

In the sequel, we present an algorithm for the computation of d based on the algorithm presented in DASGUPTA *et al.* [79] for the computation of the length of the longest path in a directed acyclic graph (DAG). This algorithm was also presented in TOMOLA *et al.* [77] and is adapted in this work to the synchronous diagnosis case.

Algorithm 3.3 *Computation of d.*

Input: G_V^{SD}.

Output: d.

1: *Create the graph $\overline{G}_V^{SD}$ by eliminating from G_V^{SD} all states that have label N and their related transitions.*

2: *Find all strongly connected components of $\overline{G}_V^{SD}$.*

3: *Obtain the acyclic graph $G_{dag} = (Q_{dag}, \Sigma_{dag}, f_{dag}, q_{0,dag})$, where $\Sigma_{dag} = \cup_{k=1}^{r} \Sigma_{N_k}^R \cup \Sigma$, from $\overline{G}_V^{SD}$ by shrinking each strongly connected component to a single state as it is done in YOO and GARCIA [80].*

4: $(v_1, v_2, \ldots, v_\eta) \leftarrow$ *Topological Sort*(G_{dag})*, where* $v_j \in Q_{dag}$*, for* $j = 1, \ldots, \eta$*, and* $\eta = |Q_{dag}|$.

5: Define the weight function $\rho : Q_{dag} \times Q_{dag} \to \{0, 1\}$*, where*

$$\rho(v_i, v_j) \leftarrow \begin{cases} 1, & \text{if } \exists \sigma \in \Sigma \text{ such that } f_{dag}(v_i, \sigma) = v_j, \\ 0, & \text{otherwise.} \end{cases}$$

6: For $j = 1, \ldots, \eta$*:*

$$l(v_j) \leftarrow \begin{cases} max\{l(v_i) + \rho(v_i, v_j) : (\exists \sigma \in \Sigma_{dag})(f_{dag}(v_i, \sigma) = v_j)\}, \\ 0, \text{if } \nexists (v_i, \sigma) \in Q_{dag} \times \Sigma_{dag} \text{ such that } (f_{dag}(v_i, \sigma) = v_j). \end{cases}$$

7: $d \leftarrow max_{j \in \{1, \ldots, \eta\}} l(v_j)$.

It is important to remark that a topological sort of a DAG is carried out in Algorithm 3.3 for the computation of d. The Topological Sort Algorithm returns the linked list of vertices of a DAG G, such that if G has an edge (u, v), then u appears before v in the ordering [79, 81].

Since the delay bound is the maximum number of events that the system can execute after the occurrence of the failure event σ_f, and until σ_f is detected, for its computation it is necessary to consider only the traces of G_V^{SD} after the occurrence of the failure event σ_f. Therefore, in Step 1 of Algorithm 3.3, the graph $\overline{G}_V^{SD}$, obtained from G_V^{SD}, is formed only with the states of G_V^{SD} reached after the occurrence of the failure event σ_f.

Notice that, in order to compute the maximum delay bound for synchronous diagnosis, the system must be synchronous diagnosable, and thus, according to Theorem 3.2, the verifier G_V^{SD} does not have any cyclic path with one of the events in the path belonging to Σ. However, the verifier G_V^{SD} can have cyclic paths formed with transitions that are labeled with renamed events. In order to eliminate the cyclic paths of $\overline{G}_V^{SD}$ it is necessary to shrink all its strongly connected components, obtaining the directed acyclic graph G_{dag}. When the strongly connected component is shrunk into one vertex, all transitions that reach or leave the strongly connected component will reach or leave the vertex. This procedure is performed in Steps 2 and 3 of Algorithm 3.3.

The Topological Sort of G_{dag} is carried out in Step 4 of Algorithm 3.3. Next, in Step 5, a weight function ρ is introduced to assign weight zero to the transitions of G_{dag} labeled with renamed events, and weight one to the transitions of G_{dag} labeled with events from Σ. Finally, in Steps 6 and 7, the number of transitions labeled with events in Σ of the longest path in the weighted acyclic graph G_{dag}, d, is computed.

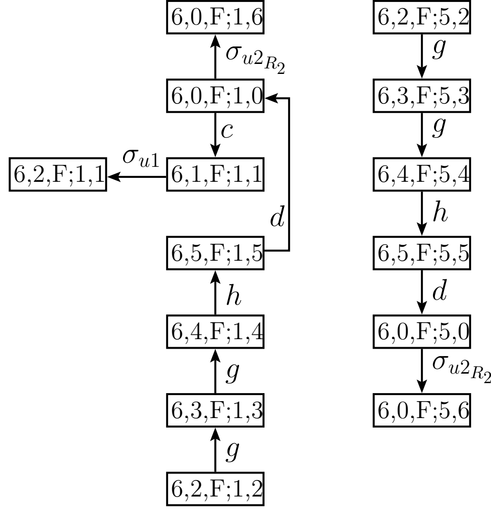


Figure 3.10: Graph $\overline{G}_V^{SD} = G_{dag}$ of Example 3.3.

After the computation of d, the delay bound for synchronous diagnosis can be computed as

$$z^* = d + 1, \tag{3.15}$$

since, for obtaining the delay bound z^*, the occurrence of the event that leads to the synchronous diagnosis of the failure event must be counted.

We present in the sequel an example to illustrate the use of Algorithm 3.3 for the computation of delay bound z^* for synchronous diagnosis.

Example 3.3 *Consider again the plant $G = G_1 \| G_2$, whose components G_1 and G_2 are shown in Figures 3.5(a) and 3.5(b), respectively. As concluded in Example 3.2, L is synchronously diagnosable with respect to L_{N_1}, L_{N_2}, $P^o_{1,o} : \Sigma_o^\star \rightarrow \Sigma_{1,o}^\star$, $P^o_{2,o} : \Sigma_o^\star \rightarrow \Sigma_{2,o}^\star$, $P_{1,o} : \Sigma^\star \rightarrow \Sigma_{1,o}^\star$, $P_{2,o} : \Sigma^\star \rightarrow \Sigma_{2,o}^\star$, $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$, and Σ_f. Thus, the maximum number of events d that can be generated after the occurrence of the failure event σ_f, for which there exists a failure trace st and a nonfailure trace $P_o(\omega)$ with the same observation, can be computed following the steps of Algorithm 3.3. In order to do so, automata $\overline{G}_V^{SD}$ and G_{dag} are computed from G_V^{SD}. In this example, $\overline{G}_V^{SD} = G_{dag}$, and is depicted in Figure 3.10. In Step 4 of Algorithm 3.3, a Topological Sort is carried out using the graph G_{dag}, which results in the graph presented in Figure 3.11. In Steps 5 and 6, the weighting functions ρ and l are computed, whose result is depicted in Figure 3.12. Finally, d is computed in Step 7, which results in $d = 6$ and, therefore, the maximum delay for synchronous diagnosis of the system $G = G_1 \| G_2$ is $z^* = 7$.*

It is important to remark that, in this example, the delay bound for the centralized diagnosis considering a monolithic diagnoser is also $z^* = 7$. This shows that, de-

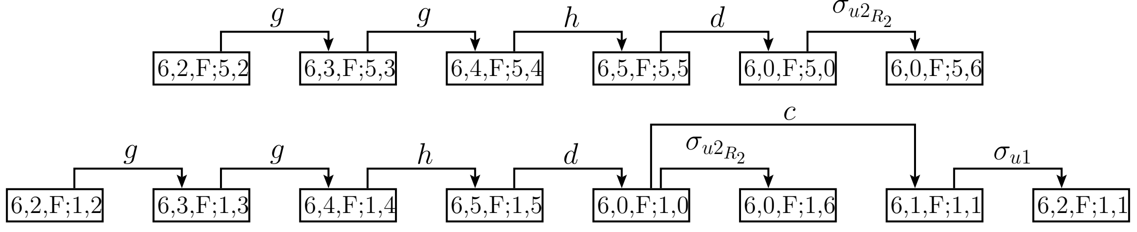


Figure 3.11: Graph G_{dag} of Example 3.3 topologically sorted.

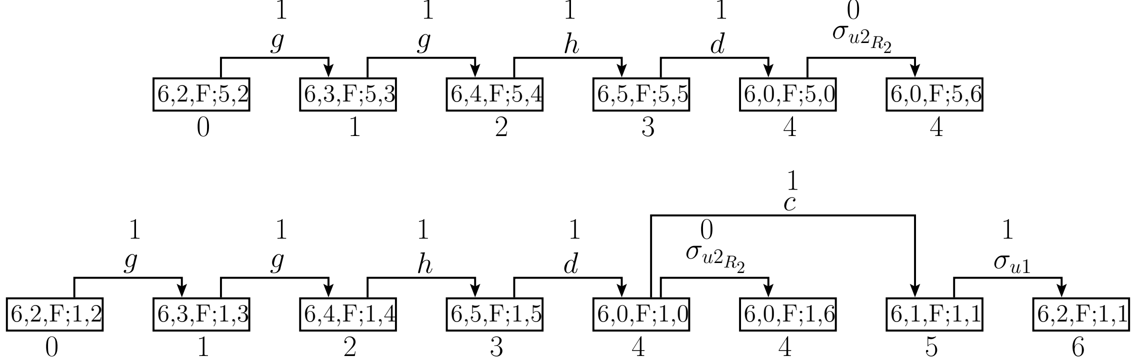


Figure 3.12: Graph G_{dag} of Example 3.3 topologically sorted with the value of the weighting functions $\rho(v_i, v_j)$ (above the edges) and $l(v_j)$ (below the vertices).

pending on the system, the failure event can be diagnosed with the same delay bound than using the monolithic approach, even with the growth of the observed nonfailure language for synchronous diagnosis.

3.3.1 Complexity analysis

The computational complexity of Algorithm 3.3 is polynomial in the size of the plant model G. In order to verify this fact, let us compute the complexity of each step of Algorithm 3.3.

In Step 1 of Algorithm 3.3, automaton $\overline{G}_V^{SD}$ is computed by eliminating from G_V^{SD} all states that are not reached after the occurrence of the failure event σ_f, and their related transitions. Thus, the maximum number of states and transitions of $\overline{G}_V^{SD}$ is bounded by the number of states and transitions of G_V^{SD}. Moreover, Step 1 of Algorithm 3.3 can be performed in linear time with respect to the size of G_V^{SD}.

In Steps 2 and 3 of Algorithm 3.3, the strongly connected components of $\overline{G}_V^{SD}$ are computed and eliminated, generating automaton G_{dag}. The search for strongly connected components in a directed acyclic graph can be done in linear time in the number of vertices and edges of the graph [81]. The number of states and edges of G_{dag} is, in the worst case, equal to the number of states and transitions of $\overline{G}_V^{SD}$, and G_{dag} can be computed in linear time in the size of $\overline{G}_V^{SD}$. The Topological Sort used

in Step 4 can also be carried out in linear time with respect to the size of G_{dag} [81]. Finally, the search for the longest weighted path of G_{dag}, presented in Step 6, can also be done in linear time in the size of G_{dag}.

The overall computational complexity of Algorithm 3.3 is equal to the complexity of the synchronous diagnosability verifier, *i.e.*, $O(r \times (\prod_{k=1}^{r} |Q_k|) \times |Q| \times |\Sigma|)$, where r is the total number of system components. Since, for the computation of z^* it is necessary to compute d using Algorithm 3.3, the complexity for the computation of the delay bound for synchronous diagnosis z^* is $O(r \times (\prod_{k=1}^{r} |Q_k|) \times |Q| \times |\Sigma|)$.

In the next section, we present a Petri net diagnoser that synchronizes the state estimate of the nonfailure behavior of the system modules on their observed events.

3.4 Synchronized Petri net diagnoser

The diagnosis method proposed in this work relies on the computation of a diagnoser that is capable of estimating the states of the system modules and synchronize the occurrence of observable events in these modules. In order to do so, it is first necessary to construct an online observer for each module that provides its current state estimate when an event is observed. The synchronization of the modules is naturally achieved by implementing the state observers running in parallel.

In Section 2.4.1, we present a state observer Petri net $\mathcal{N}_{SO}$ for DESs modeled by finite state automata [26]. This Petri net is binary and provides the state estimate of an automaton after the occurrence of an observable event. If an event σ_o is observed, and σ_o is not feasible in the current state estimate of the nonfailure behavior of the system, then all tokens of $\mathcal{N}_{SO}$ are removed which implies in the detection of the occurrence of the failure event.

The diagnosis scheme proposed in this work is based on the construction of Petri net state observers $\mathcal{N}_{SO_k}$, for $k = 1, \ldots, r$, of the nonfailure behavior automaton models of all components of the system G_{N_k}. Thus, as in CABRAL *et al.* [26], if an event σ_o is observed and σ_o is not feasible in the current state estimate of $\mathcal{N}_{SO_k}$, for $k \in \{1, \ldots, r\}$, then all tokens of $\mathcal{N}_{SO_k}$ are removed. In order to obtain the synchronized Petri net diagnoser $\mathcal{N}_D$, it is necessary to implement the Petri net state observers $\mathcal{N}_{SO_k}$ running in parallel, and to construct a failure detection logic that is capable of detecting the occurrence of the failure event when all tokens from at least one $\mathcal{N}_{SO_k}$ are removed, forming a unique Petri net diagnoser $\mathcal{N}_D$.

In the sequel, we present Algorithm 3.4 for the computation of the SPND, $\mathcal{N}_D$.

Algorithm 3.4 *Synchronized Petri net diagnoser.*

Input: *Automata* G_{N_k}, *for* $k = 1, \ldots, r$.
Output: *Synchronized Petri net diagnoser* $\mathcal{N}_D$.

1: Compute the state observers Petri nets $\mathcal{N}_{SO_k} = (P_{SO_k}, T_{SO_k}, Pre_{SO_k}, Post_{SO_k}, x_{0,SO_k}, \Sigma_{o_k}, l_{SO_k})$*, for* $k = 1, \ldots, r$*, according to Algorithm 2.5 [26], that provides the online state estimate of the nonfailure behavior of the system modules* G_{N_k}*.*

2: Compute the Petri nets $\mathcal{N}_{D_k} = (P_{D_k}, T_{D_k}, Pre_{D_k}, Post_{D_k}, In_{D_k}, x_{0,D_k}, \Sigma_{o_k}, l_{SO_k})$*, for* $k = 1, \ldots, r$*, where* $In_{D_k} : P_{D_k} \times T_{D_k} \to \{0, 1\}$ *denotes the function of inhibitor arcs [4], as follows:*

2.1: Add to $\mathcal{N}_{SO_k}$ *a transition* t_{f_k} *labeled with the always occurring event* λ*. Define* $T_{D_k} \leftarrow T_{SO_k} \cup \{t_{f_k}\}$*.*

2.2: Add to $\mathcal{N}_{SO_k}$ *a place* p_{N_k}*, and define* $Pre_{D_k}(p_{N_k}, t_{f_k}) \leftarrow 1$*. Set* $x_{0,D_k}(p_{N_k}) \leftarrow 1$*, and define* $P_{D_k} \leftarrow P_{SO_k} \cup \{p_{N_k}\}$*.*

2.3: Define $In_{D_k}(p_{D_k}, t_{f_k}) \leftarrow 1$ *and* $In_{D_k}(p_{D_k}, t_{SO_k}) \leftarrow 0$*,* $\forall p_{D_k} \in P_{D_k}$ *and* $\forall t_{SO_k} \in T_{SO_k}$*.*

3: Compute the synchronized Petri net diagnoser $\mathcal{N}_D = (P_D, T_D, Pre_D, Post_D, In_D, x_{0,D}, \Sigma_o, l_{SO})$*, as follows:*

3.1: Form a unique Petri net by grouping all Petri nets $\mathcal{N}_{D_k}$*, for* $k = 1, \ldots, r$*.*

3.2: Add a place p_F *and define* $Post_D(t_{f_k}, p_F) = 1$*, for* $k = 1, \ldots, r$*. Set* $x_{0,D}(p_F) \leftarrow 0$*.*

In order to prove that the Synchronized Petri net diagnoser $\mathcal{N}_D$, obtained from Algorithm 3.4, can be used for synchronous diagnosis, we first introduce the following lemma that shows that the state of the Petri net state observer $\mathcal{N}_{SO_k}$, reached after the observation of a trace $\nu_k \in \Sigma_{k,o}^{\star}$, provides the correct state estimate of G_{N_k}.

Lemma 3.4 *Let* $\underline{x}_{SO_k}$ *denote the state of* $\mathcal{N}_{SO_k}$ *reached after the observation of a trace* ν_k*, and let* q_{obs_k} *denote the state of* $Obs(G_{N_k}, \Sigma_{k,o})$ *reached after the observation of* ν_k*. Then, there exists a place* $p^i_{SO_k} \in P_{SO_k}$ *such that* $x_{SO_k}(p^i_{SO_k}) = 1$ *if and only if* $p^i_{SO_k}$ *is associated with a coordinate* $q^i_{N_k}$ *in* q_{obs_k}*.*

Proof. In CABRAL *et al.* [26] it is presented a method to construct the state observer Petri net $\mathcal{N}_{SO}$ from an automaton G, and it is shown that $\mathcal{N}_{SO}$ provides correctly the state estimate of automaton G. Therefore, the Petri net state observer $\mathcal{N}_{SO_k}$, obtained by applying the same method to automaton G_{N_k}, provides the estate estimate of G_{N_k}. ■

The result presented in Lemma 3.4 leads to the following theorem.

Theorem 3.3 *Let L_F be the language marked by G_F, which models the failure behavior of the system model $G = \|_{k=1}^{r} G_k$, and let $L_{N_a} = \bigcap_{k=1}^{r} P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))$. Consider language $L_a = L_{N_a} \cup \overline{L}_F$, and assume that L is synchronously diagnosable with respect to L_{N_k}, $P_{k,o}^{o} : \Sigma_o^{\star} \rightarrow \Sigma_{k,o}^{\star}$, $P_{k,o} : \Sigma^{\star} \rightarrow \Sigma_{k,o}^{\star}$, for $k = 1, \ldots, r$, $P_o : \Sigma^{\star} \rightarrow \Sigma_o^{\star}$, and Σ_f. Let $s \in L_a \setminus L_{N_a}$ be an arbitrarily long trace, such that $\forall \omega \in L_a$ satisfying $P_o(\omega) = P_o(s)$, $\omega \in L_a \setminus L_{N_a}$. Then, the number of tokens in place p_F of $\mathcal{N}_D$, after the observation of the trace $P_o(s)$, is one.*

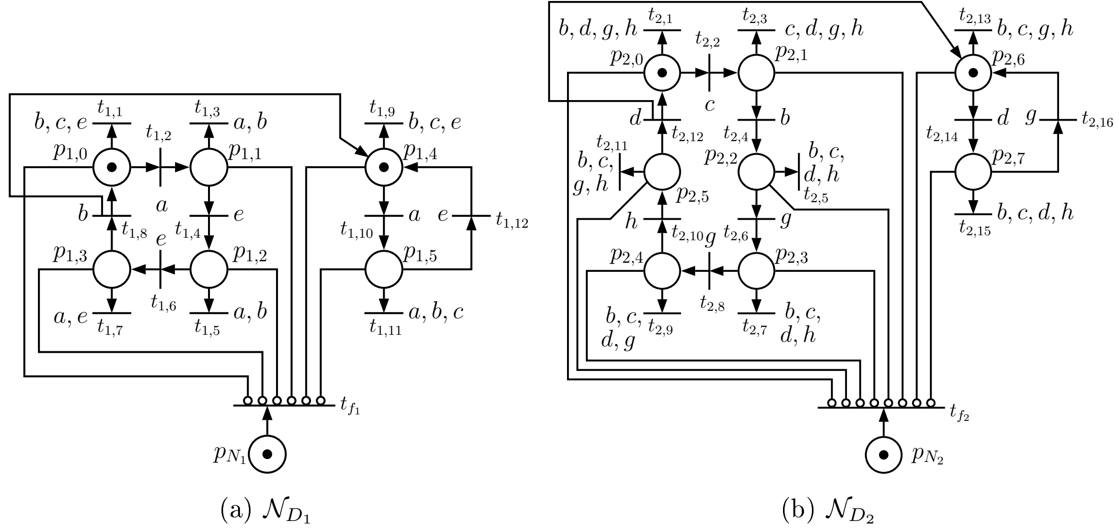
Proof. If L is synchronously diagnosable and the system generates an unambiguous trace s, then $P_o(s) \notin L_{N_a}$, where $L_{N_a} = \cap_{k=1}^{r} P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))$, according to Definition 3.1 of synchronous diagnosability. Since $P_o(s) \notin \cap_{k=1}^{r} P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))$, then, $\exists k \in \{1, \ldots, r\}$ such that $P_o(s) \notin P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))$. Since $P_o(s) \notin P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))$ for $k \in \{1, \ldots, r\}$, then $P_{k,o}^{o}(P_o(s)) \notin P_{k,o}^{o}(P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k})))$ for $k \in \{1, \ldots, r\}$, due to the definitions of $P_{k,o}^{o} : \Sigma_o^{\star} \rightarrow \Sigma_{k,o}^{\star}$ and $P_o : \Sigma^{\star} \rightarrow \Sigma_o^{\star}$. Notice that $P_{k,o}^{o}(P_o(s)) = P_{k,o}(s)$ and $P_{k,o}^{o}(P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))) = P_{k,o}(L_{N_k})$, thus, $\exists k \in \{1, \ldots, r\}$ such that $P_{k,o}(s) \notin P_{k,o}(L_{N_k})$. Therefore, since, according to Lemma 3.4, $\mathcal{N}_{SO_k}$ provides the state estimate of G_{N_k}, after the occurrence of s, and $\exists k \in \{1, \ldots, r\}$ such that $P_{k,o}(s) \notin P_{k,o}(L_{N_k})$ then at least one state observer Petri net will lose all its tokens, which will enable transition t_{f_k}, labeled with the always occurring event λ. When transition t_{f_k} fires, it adds a token in place F, indicating the occurrence of a failure event. ■

Remark 3.2 *It is important to remark that the computational complexity for the construction of the SPND is polynomial in the number of states and transitions of G_{N_k}, i.e., the SPND can be obtained in polynomial time in the number of states and transitions of the modules of the system, avoiding the computation of the global system model for implementation of the diagnoser.*

In the following, we present an example to illustrate the construction and the diagnosis procedure of the synchronous Petri net diagnoser.

Example 3.4 *Consider again the global plant model $G = G_1 \| G_2$, where G_1 and G_2 are depicted in Figures 3.2(a) and 3.2(b), respectively. According to Algorithm 3.4, the first step for the computation of the SPND is the computation of the state observer Petri nets $\mathcal{N}_{SO_1}$ and $\mathcal{N}_{SO_2}$. In the sequel, Petri nets $\mathcal{N}_{D_1}$ and $\mathcal{N}_{D_2}$, shown in Figures 3.13(a) and 3.13(b), respectively, are computed. Finally, the SPND is obtained by grouping $\mathcal{N}_{D_1}$ and $\mathcal{N}_{D_2}$, and adding a place p_F that receives a token only when the failure event is diagnosed, indicating its occurrence. The SPND $\mathcal{N}_D$ is presented in Figure 3.14.*

Let us now show how online diagnosis is carried out by using $\mathcal{N}_D$. Suppose that the failure trace $s_1 = a\sigma_f(a)^z \in L \setminus L_N$ for $z \in \mathbb{N}$ has been executed by the system.



(a) $\mathcal{N}_{D_1}$ (b) $\mathcal{N}_{D_2}$

Figure 3.13: Petri nets $\mathcal{N}_{D_1}$ and $\mathcal{N}_{D_2}$ of Example 3.4.

Then, the trace observed in the first module is $P_{1,o}(s_1) = aa^z$ *and in the second module is* $P_{2,o}(s_1) = \varepsilon$*. Since in the initial marking of* $\mathcal{N}_{D_1}$ *places* $p_{1,0}$ *and* $p_{1,4}$ *have tokens, transitions* $t_{1,2}$ *and* $t_{2,14}$ *will fire after the observation of event* a*, which will remove the tokens from places* $p_{1,0}$ *and* $p_{1,4}$ *and add tokens to places* $p_{1,1}$ *and* $p_{1,5}$*. When the second event* a *is observed, transitions* $t_{1,3}$ *and* $t_{2,14}$ *will fire, removing the tokens from places* $p_{1,1}$ *and* $p_{1,5}$*. At this point, transition* t_{f_1} *is enabled and fires, indicating that the failure has occurred and has been diagnosed.*

Now, suppose that the failure trace $s_2 = a\sigma_f(c\sigma_{u1}gghd)^z \in L \setminus L_N$ *for* $z \in \mathbb{N}$ *has been executed by the system. Then, the trace observed in the first module is* $P_{1,o}(s_2) = a(c)^z$ *and in the second module is* $P_{2,o}(s_2) = (cgghd)^z$*. In the initial marking of* $\mathcal{N}_D$ *of Figure 3.14, places* $p_{1,0}$*,* $p_{1,4}$*,* $p_{2,0}$ *and* $p_{2,6}$ *have tokens. When event* a *is observed, transitions* $t_{1,2}$ *and* $t_{1,10}$ *fire, removing tokens from places* $p_{1,0}$ *and* $p_{1,4}$*, and adding tokens to places* $p_{1,1}$ *and* $p_{1,5}$*. When event* c *is observed, transitions* $t_{1,11}$*,* $t_{2,2}$*, and* $t_{2,13}$ *fire, removing tokens from the places* $p_{1,5}$*,* $p_{2,0}$*, and* $p_{2,6}$ *and adding a token to place* $p_{2,1}$*. The places that have tokens after the trace* $a\sigma_f c$ *has been executed by the system are* $p_{1,1}$ *and* $p_{2,1}$*. After the observation of event* g*, transition* $t_{2,3}$ *fires, removing the token from place* $p_{2,1}$*. At this point, transition* t_{f_2} *is enabled and, since it is labeled with the always occurring event* λ*, it fires removing a token from place* p_{N_2} *and adding a token to place* p_F*, which diagnoses the failure event* σ_f*.*

It is important to notice that the two modules of the system are important for the diagnosis of the failure event σ_f*. For example, the failure trace* s_2 *is diagnosed from the Petri net* $\mathcal{N}_{D_2}$*, which is built from the nonfailure model of component* G_2*,* G_{N_2}*. Notice that the failure event* σ_f *is not even modeled in automaton* G_2*.*

Remark 3.3 *It is important to remark that the computation of the state estimate*

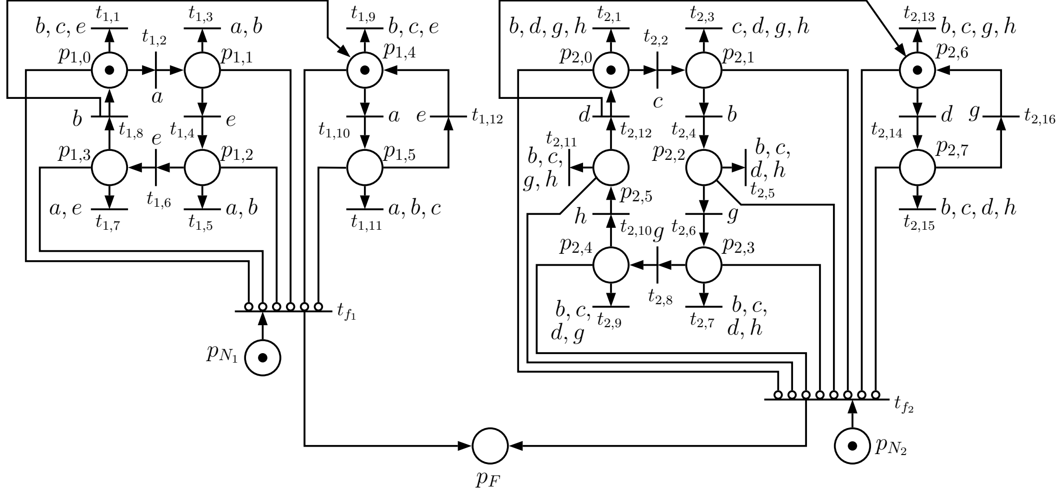


Figure 3.14: Synchronized Petri net diagnoser $\mathcal{N}_D$ of Example 3.4.

of the synchronous Petri net diagnoser, after the occurrence of an observable event $\sigma_o \in \Sigma_o$ can be carried out in two steps: (i) identify the transitions labeled with the event σ_o; (ii) fire these transitions and update the marking of the SPND. This procedure has linear computational complexity with respect to the size of the SPND. Since, according to Remark 3.2, $\mathcal{N}_D$ can be obtained in polynomial time with respect to the number of states and transitions of automata G_{N_k}, then the computational complexity of the diagnosis procedure is also polynomial with respect to the size of automata G_{N_k}.

The synchronous diagnosis method was implemented in a mechatronic plant located at the Control and Automation Laboratory (LCA) of the Federal University of Rio de Janeiro (UFRJ). In the next section, we present the functioning of this mechatronic system and the model obtained for failure diagnosis.

3.5 Synchronized Petri net diagnoser for an automated system

In this section, we present the design of the synchronized Petri net diagnoser for a mechatronic system [66]. In order to do so, we first present the controlled plant.

3.5.1 Case study system

The controlled plant is a cube assembly mechatronic system of the manufacturer Christiani [82] installed at the Control and Automation Laboratory of the Federal University of Rio de Janeiro. This mechatronic system consists of three modules:

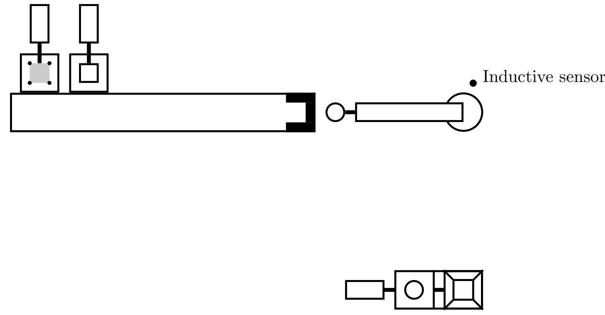


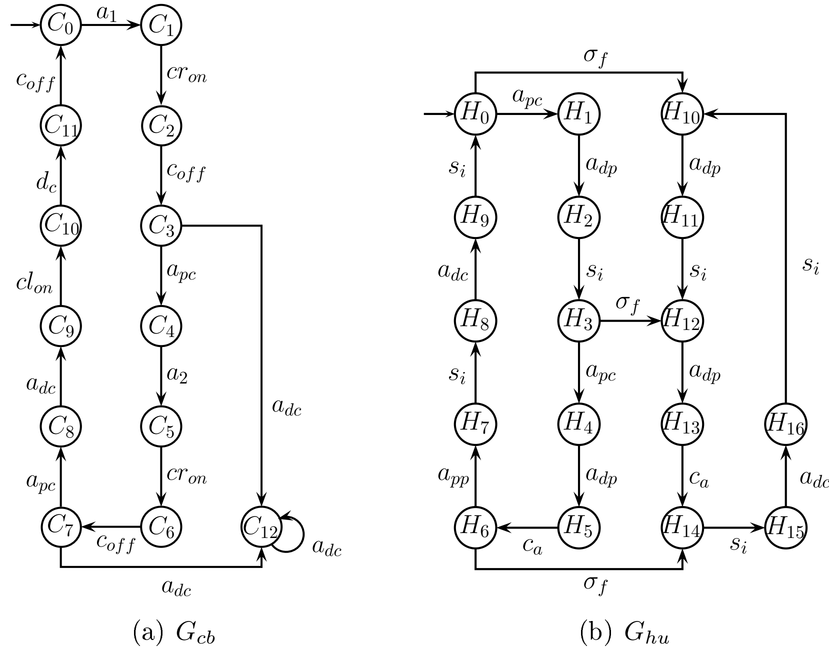
Figure 3.15: Schematics of the conveyor belt and the handling unit considered in the case study.

(i) a conveyor belt with a sensor testing unit that can be fed with plastic or metallic cube halves; (ii) a handling unit composed of a robotic arm, which has a pneumatic mechanism that activates a suction cup in order to pick up, transport and deliver pieces to a press used to assemble two halves of a cube; and (iii) a magazine unit that stores the assembled cube in a shelf unit with 28 available storage positions. In this work, only the first two modules are considered, *i.e.*, the conveyor belt and the handling unit. We show a schematics of the case study system in Figure 3.15, where the conveyor belt and the handling unit can be seen in an aerial view.

We have designed the automated system to deliver two cube halves to the press, then the press assembles the cube halves into one cube and then the cube is discarded. The behavior of the controlled system is as follows: first, the conveyor belt is fed with a metallic cube half that is delivered to the handling unit. The robotic arm allocates the metallic cube half in the press and waits for a plastic cube half. Then, a plastic cube half is delivered to the conveyor belt and it is transported to the handling unit. The plastic cube half is also delivered to the press by the robotic arm, and, after that, the cube halves are assembled into one cube by the press. After that, the robotic arm removes the cube from the press and delivers it to the end of the conveyor belt, which is switched on in reverse until the cube is discarded from the system. Then, the conveyor belt is switched off and the system is ready to start the process again. Notice that only the conveyor belt and the handling unit were used in this work.

3.5.2 Modeling the controlled plant

The controlled plant has been modeled with a view to implementing the synchronized Petri net diagnoser. In order to do so, we have considered that the global system is composed of two modules: (i) the conveyor belt, and (ii) the handling unit. The automaton models of the conveyor belt G_{cb}, and the handling unit G_{hu} can be seen in Figures 3.16(a) and 3.16(b), respectively, and the global plant model, G_p, is given



(a) G_{cb} (b) G_{hu}

Figure 3.16: Automaton G_{cb} that models the conveyor belt (a); and automaton G_{hu} that models the handling unit (b).

by $G_p = G_{cb} \| G_{hu}$.

The initial state of automaton G_{cb} is C_0 which represents that the conveyor belt is turned off and there are no cube halves on it. A new piece, consisting of a cube half, is delivered to the conveyor belt, modeled as event a_1, and G_{cb} reaches state C_1, which means that the conveyor belt is turned off and the first cube half is on it. Then, the conveyor belt is switched on to the direction of the handling unit, modeled by event cr_{on}, which generates the change from state C_1 to state C_2. When the cube half reaches the end of the conveyor belt, event c_{off} occurs, and the system evolves to state C_3, indicating that the conveyor belt is switched off. When the cube half is removed from the conveyor belt by the robotic arm, event a_{pc} occurs, and the system evolves to state C_4. In state C_4, the conveyor belt is turned off waiting for the arrival of the second cube half. When the second cube half is delivered to the conveyor belt, event a_2 occurs and G_{cb} evolves to state C_5. The system repeats the same behavior with the second cube half until it reaches state C_8, indicating that the second cube half has been removed from the conveyor belt and the system is waiting for the robotic arm to deliver an assembled cube. When the robotic arm delivers the cube to the end of the conveyor belt, event a_{dc} occurs, and the system evolves to state C_9. After the cube is delivered, the conveyor belt is switched on in the opposite direction, modeled by event cl_{on}, and G_{cb} reaches state C_{10}. G_{cb} stays in state C_{10} until the cube is discarded from the system, modeled by event d_c, which makes G_{cb} reach state C_{11}. After that, the conveyor belt is switched off when event c_{off} occurs, and G_{cb} returns to its initial state. In this system, we have considered that a failure event can

occur in the robotic arm. This failure, represented by event σ_f, models the suction cup malfunctioning of the robotic arm. As a consequence, pieces cannot be removed from the end of the conveyor belt. Thus, after the occurrence of the failure event, event a_{pc} cannot occur anymore, and event a_{dc} can occur indefinitely, indicating that the robotic arm will continue to try, unsuccessfully, to pick up pieces at the end of the conveyor belt and deliver assembled cubes. This behavior is modeled by transitions (C_3, a_{dc}, C_{12}), (C_7, a_{dc}, C_{12}), and (C_{12}, a_{dc}, C_{12}) in G_{cb}, indicating that a piece is at the end of the conveyor belt and only event a_{dc} can occur again.

The initial state of the handling unit model G_{hu} is H_0, which represents that the robotic arm is aligned with the conveyor belt. If there is a cube half in the end of the conveyor belt, event a_{pc} occurs and G_{hu} reaches state H_1, indicating that the robotic arm removed the first cube half from the conveyor belt and is transporting it to the press. When the robotic arm delivers the first cube half to the press, event a_{dp} occurs and the system evolves to state H_2.

The robotic arm is equipped with a high speed counter that is triggered when it starts to turn. When the high speed counter reaches a previously known value that represents a given angular position, the robotic arm stops. In order to avoid positioning errors, after delivering a piece to the press or removing an assembled cube from the press, the robotic arm must rotate to a position where an inductive sensor is activated and the high speed counter is reseted. This process is modeled by event s_i such that, when event s_i occurs, the process of delivering a piece to the press or removing a cube from the press is completed and the robotic arm is ready to remove a new piece from the conveyor belt or deliver the assembled cube to the conveyor belt. Thus, after s_i occurs, G_{hu} evolves from state H_2 to state H_3 and the robotic arm is ready to remove the second cube half from the conveyor belt. The process is repeated for the second cube half until a cube is assembled by the press, modeled by event c_a, reaching state H_6 in G_{hu}. At this moment, the robotic arm removes the cube from the press, modeled by event a_{pp} and starts to transport it to the conveyor belt, represented by state H_7. After event s_i occurs, the robotic arm aligns with the conveyor belt and delivers the cube, modeled by event a_{dc}. Finally, after a new occurrence of event s_i, the handling unit reaches its initial state.

If the failure event σ_f occurs, the robotic arm cannot pick up cube halves at the conveyor belt or cubes at the press. As a consequence, events a_{pc} and a_{pp} cannot occur in G_{hu} after the failure event. Since there are no sensors that can indicate the presence of a piece in the handling unit, then, if a failure occurs, the robotic arm and the press will perform their cyclic behavior without any cube halves or cubes been transported or assembled.

The set of events of G_{cb} and G_{hu} are $\Sigma_{cb} = \{a_1, a_2, cr_{on}, c_{off}, a_{pc}, a_{dc}, cl_{on}, d_c\}$ and $\Sigma_{hu} = \{a_{pc}, a_{dp}, s_i, c_a, a_{pp}, a_{dc}, \sigma_f\}$, respectively, where $\Sigma_{cb,o} =$

Table 3.1: States of G_{cb}.

State	Meaning
C_0	Conveyor belt switched off and there are no cube halves on it
C_1	Conveyor belt switched off and the first cube half is on it
C_2	Conveyor belt switched on with the first cube half on it
C_3	Conveyor belt switched off and the first cube half is at its end
C_4	Conveyor belt switched off waiting for the second cube half
C_5	Conveyor belt switched off with the second cube half on it
C_6	Conveyor belt switched on with the second cube half on it
C_7	Conveyor belt switched off and the second cube half is at its end
C_8	Conveyor belt switched off waiting for the assembled cube
C_9	Conveyor belt switched off with the assembled cube at its end
C_{10}	Conveyor belt switched on in the opposite direction with a cube on it
C_{11}	Conveyor belt switched on in the opposite direction with no pieces on it
C_{12}	Conveyor belt switched off with one cube half after failure

$\{a_1, a_2, cr_{on}, c_{off}, a_{dc}, cl_{on}, d_c\}$ and $\Sigma_{hu,o} = \{a_{dp}, s_i, c_a, a_{dc}\}$ are the sets of observable events of G_{cb} and G_{hu}, and $\Sigma_{cb,uo} = \{a_{pc}\}$ and $\Sigma_{hu,uo} = \{a_{pc}, a_{pp}, \sigma_f\}$ are the sets of unobservable events of G_{cb} and G_{hu}, respectively. The sets of events, observable events, and unobservable events of the plant are, respectively, $\Sigma_p = \Sigma_{cb} \cup \Sigma_{hu}$, $\Sigma_{p,o} = \Sigma_{cb,o} \cup \Sigma_{hu,o}$, and $\Sigma_{p,uo} = \Sigma_{cb,uo} \cup \Sigma_{hu,uo}$. It is important to notice that, since the synchronized Petri net diagnoser will be implemented in the same PLC as the system controller, the information of both controller commands and sensor readings contribute to the observable events set $\Sigma_{p,o}$ of G_p. We summarize the states of G_{cb} and G_{hu} in Tables 3.1 and 3.2, respectively, and the events of G_p in Table 3.3.

3.5.3 Synchronized Petri net diagnoser

In order to build the synchronized Petri net diagnoser of the system $G_p = G_{cb} \| G_{hu}$, it is first necessary to verify if G_p is synchronously diagnosable. In order to do so, following Algorithm 3.2 the first step is to build automaton G_F, whose marked language models de failure language of the system. Automaton G_F obtained from G_p is shown in Figure 3.17. Since, for the verification of synchronous diagnosability, the marked states of G_F are not relevant, the states labeled with F in G_F are not marked. For this example, automaton G_F is equal to automaton G_p except from the labels N and F. In Step 2, automata $G_{N_{cb}}$ and $G_{N_{hu}}$ are obtained and are shown in Figures 3.18(a) and 3.18(b), respectively. Notice that automaton $G_{N_{cb}}$ is different from automaton G_{cb}, even with the failure event been modeled only in automaton G_{hu}. In Step 3, automaton G_N^R is computed from automata $G_{N_{cb}}^R$ and $G_{N_{hu}}^R$ that are depicted in Figures 3.19(a) and 3.19(b), respectively. Finally, the verifier G_V^{SD} is computed in Step 4. Verifier G_V^{SD} is omitted here due to the lack

Table 3.2: States of G_{hu}.

State	Meaning
H_0	Robotic arm ready to remove the first cube half
H_1	Robotic arm is transporting the first cube half to the press
H_2	Robotic arm returning to its initial position
H_3	Robotic arm ready to remove the second cube half
H_4	Robotic arm is transporting the second cube half to the press
H_5	Robotic arm waiting for the cube to be assembled by the press
H_6	Robotic arm starts the process of removing the cube
H_7	Robotic arm transporting the cube to the conveyor belt
H_8	Robotic arm aligning to the conveyor belt
H_9	Robotic arm moving to its initial position
H_{10}	Robotic arm moving to the press with its vacuum system malfunctioning
H_{11}	Robotic arm returning to its initial position with its vacuum system malfunctioning
H_{12}	Robotic arm ready to remove the second cube half with its vacuum system malfunctioning
H_{13}	Robotic arm waiting for the cube to be assembled by the press with its vacuum system malfunctioning
H_{14}	Press finishes the cube and robotic arm starts the process of removing the cube with its vacuum system malfunctioning
H_{15}	Robotic arm aligning to the conveyor belt with its vacuum system malfunctioning
H_{16}	Robotic arm moving to its initial position with its vacuum system malfunctioning

Table 3.3: Events of G_p.

Event	Meaning
a_1	First cube half arrives at the conveyor belt
a_2	Second cube half arrives at the conveyor belt
cr_{on}	The conveyor belt is switched on
c_{off}	The conveyor belt is switched off
a_{pc}	Robotic arm removes a piece from the conveyor belt
a_{dc}	Robotic arm delivers a cube to the conveyor belt
cl_{on}	Conveyor belt is switched on in the opposite direction
d_c	A cube is discarded from the system
a_{dp}	Robotic arm delivers a piece to the press
s_i	Inductive sensor is activated
c_a	Press finishes the assembling of a cube
a_{pp}	Robotic arm removes a cube from the press
σ_f	The robotic suction cup fails

of space. Automaton G_V^{SD} does not have any cycle of states satisfying Condition (3.14), and, thus, the system G_p is synchronous diagnosable.

The synchronized Petri net diagnoser $\mathcal{N}_{D_p}$ shown in Figure 3.20 is formed by the Petri nets $\mathcal{N}_{D_{cb}}$ and $\mathcal{N}_{D_{hu}}$. $\mathcal{N}_{D_p}$ is computed by following the steps of Algorithm 3.4, from the nonfailure behavior models $G_{N_{cb}}$ and $G_{N_{hu}}$. Let us now show how online diagnosis is carried out by using $\mathcal{N}_{D_p}$. Suppose that the failure trace $s = a_1 cr_{on} c_{off} \sigma_f a_{dp} s_i a_{dp} c_a s_i a_{dc}$ has been executed by the system. Before the occurrence of event a_{dc}, the places p_{C_3}, p_{C_4} and p_{H_8} have tokens and, when event a_{dc} is executed by the system, transitions $t_{1,7}$ and $t_{1,8}$ fire, removing the tokens from places p_{C_3} and p_{C_4}. At this point, the state observer Petri net $\mathcal{N}_{SO_{cb}}$ loses all its tokens, enabling transition t_{f_1} that fires, indicating that the failure event has occurred.

We also have computed the delay bound z^* for synchronous diagnosis of system G_p. By following Algorithm 3.3 and Equation (3.15), the maximum number of events that G_p can generate after the failure event σ_f until σ_f is diagnosed by $\mathcal{N}_{D_p}$ is $z^* = 15$. The delay bound for monolithic diagnosis is 12. As presented in Section 3.3, this difference is due to the growth of the nonfailure language for synchronous diagnosis compared to the monolithic diagnosis scheme. The diagnoser $\mathcal{N}_{D_p}$ can be converted into a Ladder diagram by using the method proposed in CABRAL *et al.* [26], and implemented on the same PLC where the control code of the system is implemented.

3.6 Final remarks

In this chapter, we propose a new architecture for the diagnosis of DESs, called synchronous diagnosis. This architecture is based on the construction of Petri net state observers of the nonfailure models of the components of the system. If an observable event that is not feasible in the current state estimate of the nonfailure behavior of at least one component is executed by the system, the failure event is diagnosed. We have shown that if there are unobservable events in common between the components of the system, the nonfailure language for synchronous diagnosis can be a larger set than the global nonfailure language of the system. Thus, the notion of synchronous diagnosability and an algorithm to verify this property are proposed. The synchronous diagnosability verification method has exponential complexity with the number of system components, in the worst case scenario. However, the main objective of this work is to present a method for the diagnosis of the failure event that avoids the use of the global plant model and, thus, avoids the exponential growth with the number of modules for failure diagnosis.

Since the nonfailure language for synchronous diagnosis can be a larger set that the nonfailure language of the system, the delay bound for synchronous diagnosis

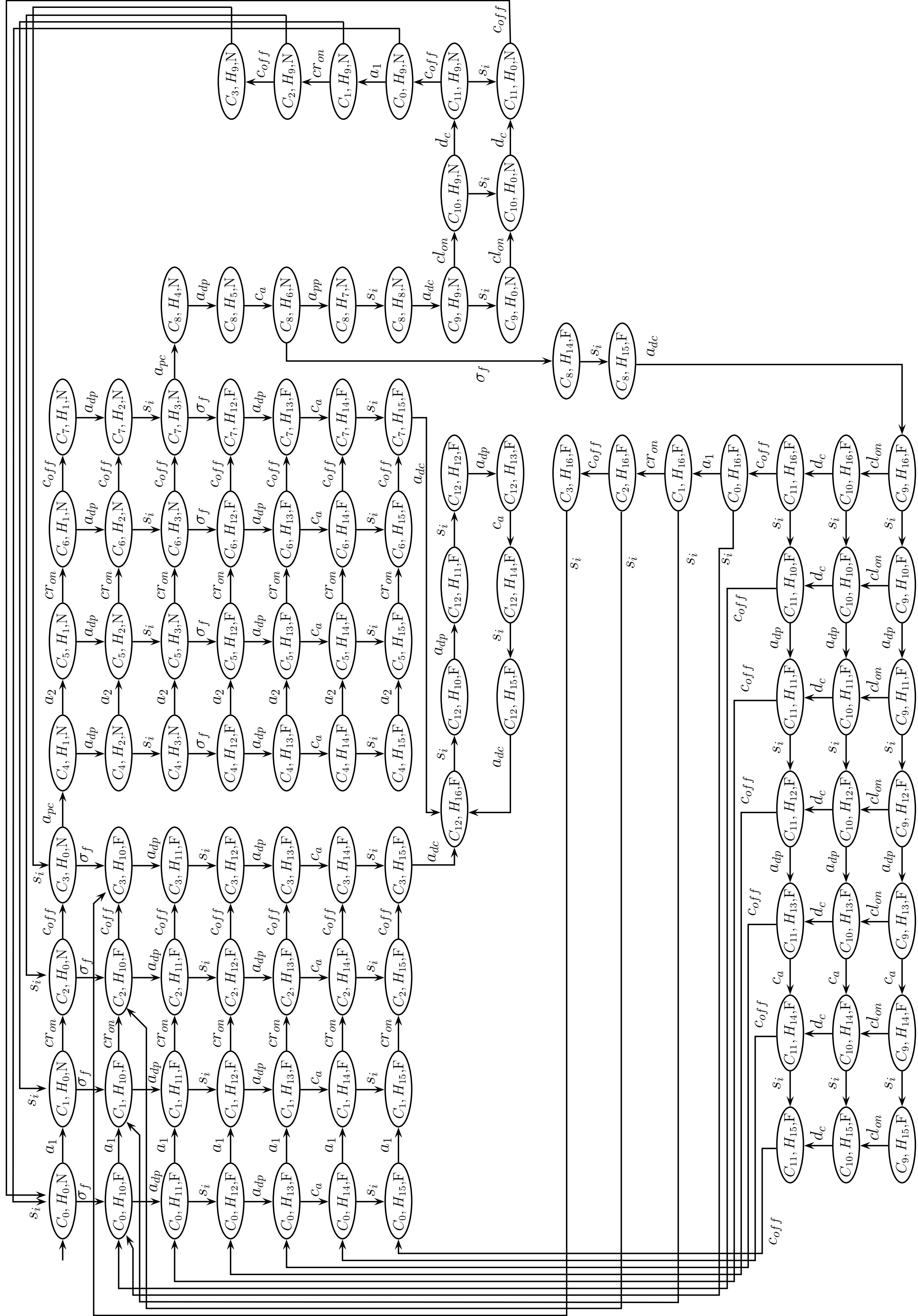
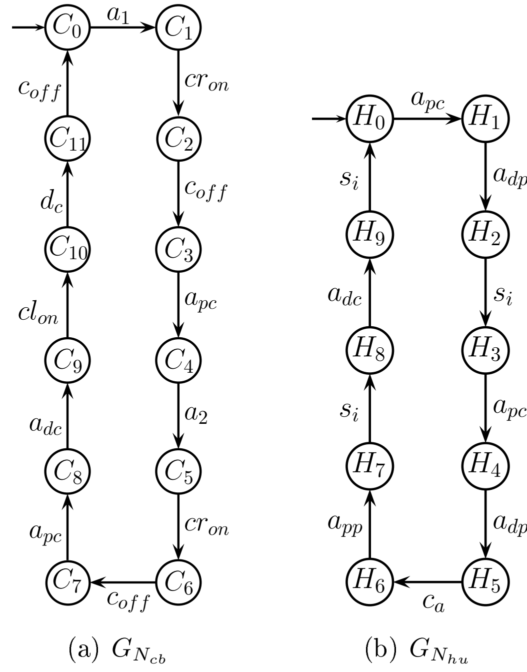
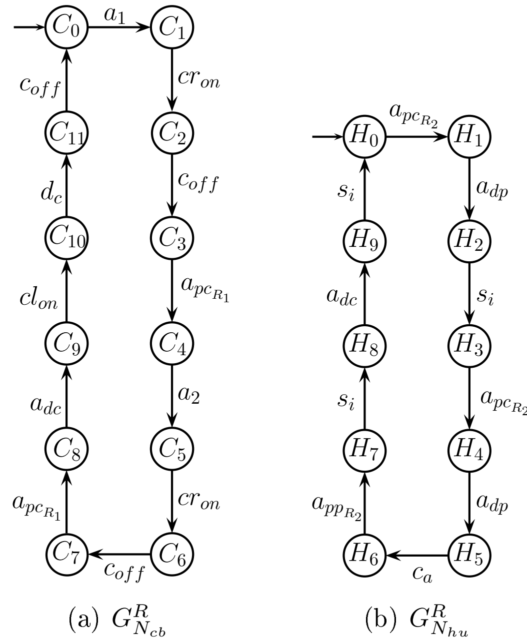


Figure 3.17: Automaton G_F obtained from G_p.



(a) $G_{N_{cb}}$ (b) $G_{N_{hu}}$

Figure 3.18: Automaton $G_{N_{cb}}$ (a); and automaton $G_{N_{hu}}$ (b).



(a) $G^R_{N_{cb}}$ (b) $G^R_{N_{hu}}$

Figure 3.19: Automaton $G^R_{N_{cb}}$ (a); and automaton $G^R_{N_{hu}}$ (b).

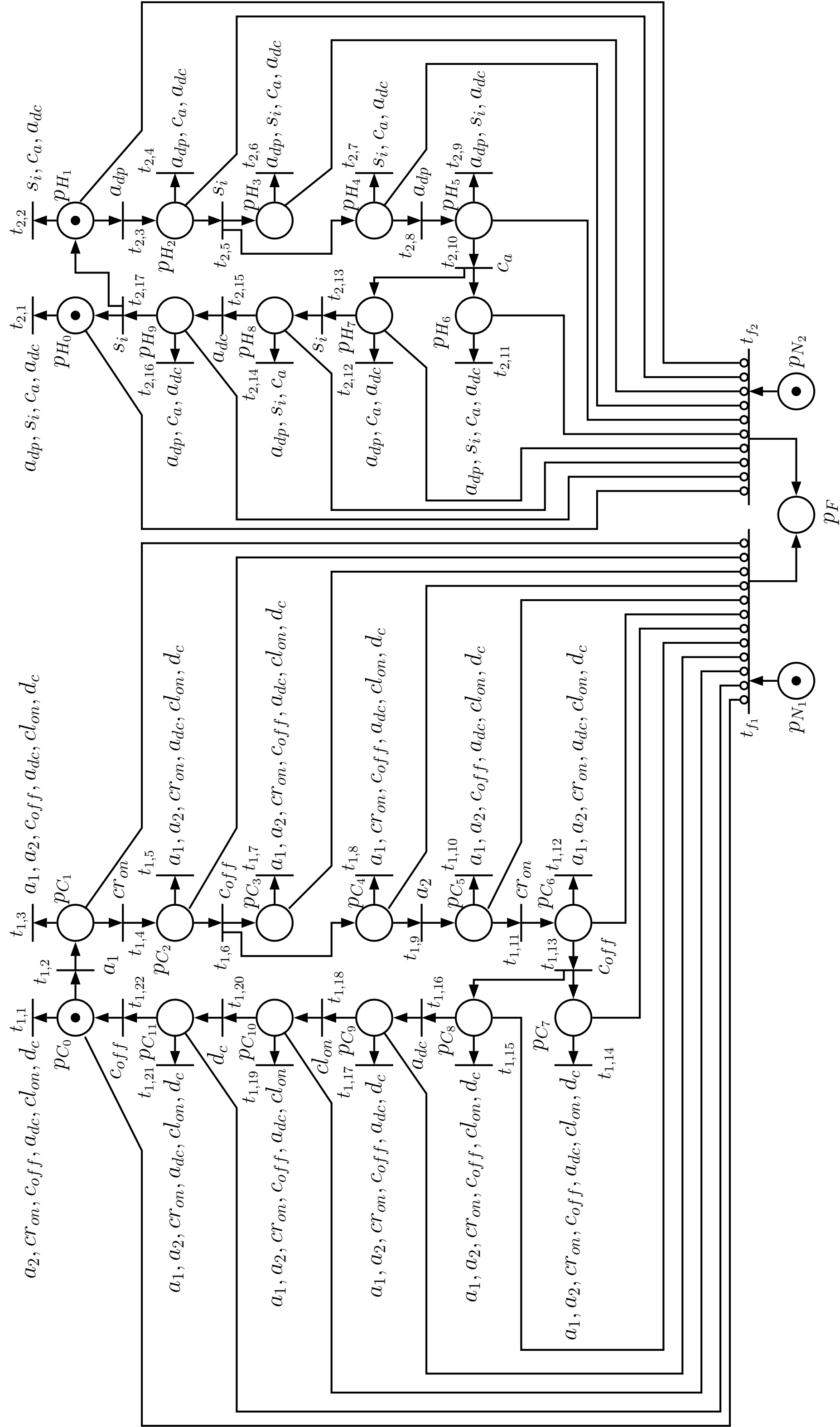


Figure 3.20: Petri net diagnoser $\mathcal{N}_{D_p}$.

can be larger than the delay bound using a monolithic diagnosis approach. In this chapter, we also presented a method for the computation of the delay bound for synchronous diagnosis.

A practical implementation of the synchronous diagnosis is also presented for a cube assembly mechatronic system. The system is composed of two modules and the modeling of the system for synchronous diagnosis is presented. The synchronous diagnoser was implemented for this system considering two different models for the same system [65, 66].

In the next chapter, we generalize the notion of synchronous diagnosis to a decentralized diagnosis scheme. We based our decentralized diagnosis scheme on Protocol 3 of DEBOUK *et al.* [17], where the local diagnosers do not communicate with each other and the failure occurrence information is sent to a coordinator, that diagnosis the failure event when at least one of the local diagnosers identifies its occurrence.

Chapter 4

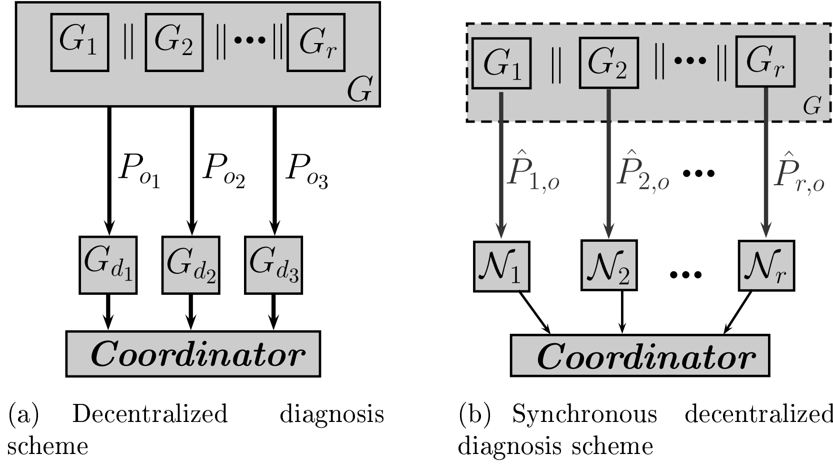
Synchronous codiagnosability of DESs

4.1 Synchronous codiagnosability

In the decentralized diagnosis scheme presented in DEBOUK *et al.* [17] each local diagnoser is constructed based on the global model G and, therefore, may grow exponentially with the number of system components. In this work, in order to avoid the exponential growth with the number of system components, we extend the synchronous diagnosis approach presented in Chapter 3 to the decentralized case. Differently from DEBOUK *et al.* [17], where the local diagnosers are constructed based on the global plant model G, in the synchronous decentralized diagnosis scheme, the local diagnosers are constructed based on the nonfailure behavior model of the system components [83, 84].

As in Chapter 3, we consider that the system G is obtained by the parallel composition of several subsystems, modeled by G_k, $k = 1, \ldots, r$. In order to consider a synchronous decentralized diagnosis scheme, we assume that each subsystem has its own set of observable events that are communicated to a local Petri net diagnoser $\mathcal{N}_k$. In Figure 4.1, we compare the decentralized diagnosis scheme with the synchronous decentralized diagnosis architecture. In the synchronous decentralized diagnosis scheme, the set of events of the system component G_k can be partitioned as $\Sigma_k = \hat{\Sigma}_{k,o} \dot{\cup} \hat{\Sigma}_{k,uo}$, where $\hat{\Sigma}_{k,o}$ and $\hat{\Sigma}_{k,uo}$ are the sets of observable and unobservable events of G_k, respectively. It is important to remark that, differently from the centralized approach presented in Chapter 3 an event can be observable to a local diagnoser $\mathcal{N}_i$ and unobservable to another local diagnoser $\mathcal{N}_j$, *i.e.*, $\hat{\Sigma}_{i,o} \cap \hat{\Sigma}_{j,uo}$ is not necessarily the empty set for $j \neq i$, $i, j \in \{1, \ldots, r\}$. Thus, $\hat{\Sigma}_{k,o} \subseteq \Sigma_{k,o}$.

Figure 4.1(b) depicts the synchronous decentralized diagnosis scheme proposed in this work. The local Petri net diagnosers $\mathcal{N}_k$ are constructed based on the nonfailure



(a) Decentralized diagnosis scheme

(b) Synchronous decentralized diagnosis scheme

Figure 4.1: Comparison between the decentralized diagnosis architecture (a); and the synchronous decentralized diagnosis architecture (b).

behavior of the system modules G_{N_k}, for $k = 1, \ldots, r$, and they infer the occurrence of failure events based on their own observations, and send the information regarding the failure occurrence to the coordinator. A failure is diagnosed when at least one local diagnoser identifies its occurrence, which occurs when an event, that is not feasible in the current state estimate of the nonfailure behavior of one component, is observed. This diagnosis scheme leads to the following definition of synchronous codiagnosability.

Definition 4.1 (Synchronous codiagnosability) *Let $G_N = \|_{k=1}^{r} G_{N_k}$, where G_{N_k} is the automaton that models the nonfailure behavior of G_k, and let L_{N_k} denote the language generated by G_{N_k}, for $k = 1, \ldots, r$, where r is the number of system components. Assume that there are r local sites with projections $\hat{P}_{k,o} : \Sigma^\star \to \hat{\Sigma}_{k,o}^\star$, $k = 1, \ldots, r$. Then, L is said to be synchronously codiagnosable with respect to L_{N_k}, $\hat{P}_{k,o}$, and Σ_f if*

$$(\exists z \in \mathbb{N})(\forall s \in L_F)(\forall st \in L_F, \|t\| \geq z) \Rightarrow$$
$$(\exists k \in \{1, 2, \ldots, r\})(\hat{P}_{k,o}(st) \notin \hat{P}_{k,o}(L_{N_k})).$$

In the sequel, we show that the synchronous codiagnosability of L with respect to L_{N_k}, $\hat{P}_{k,o} : \Sigma^\star \to \hat{\Sigma}_{k,o}^\star$, and Σ_f implies in the synchronous diagnosability of L with respect to L_{N_k}, $P_{k,o}^o : \Sigma_o^\star \to \Sigma_{k,o}^\star$, $P_{k,o} : \Sigma^\star \to \Sigma_{k,o}^\star$, for $k = 1, \ldots, r$, $P_o : \Sigma^\star \to \Sigma_o^\star$, and Σ_f.

Lemma 4.1 *Let st be a failure trace and L_{N_k} be the language generated by G_{N_k}, for $k = 1, \ldots, r$. Consider the projections $\hat{P}_{k,o} : \Sigma^\star \to \hat{\Sigma}_{k,o}^\star$, $\hat{P}_{k,o}^o : \Sigma_o^\star \to \hat{\Sigma}_{k,o}^\star$, for $k = 1, \ldots, r$, and $P_o : \Sigma^\star \to \Sigma_o^\star$, where $\Sigma_o = \cup_{k=1}^{r} \hat{\Sigma}_{k,o} = \cup_{k=1}^{r} \Sigma_{k,o}$. Then,*

$\exists k \in \{1,\ldots,r\}$ such that $\hat{P}_{k,o}(st) \notin \hat{P}_{k,o}(L_{N_k})$, if, and only if,

$$P_o(st) \notin \hat{L}_{N_a}, \tag{4.1}$$

where $\hat{L}_{N_a} = \cap_{k=1}^{r} \hat{P}_{k,o}^{o^{-1}}(\hat{P}_{k,o}(L_{N_k}))$.

Proof. ($\Rightarrow$) Let $k \in \{1,\ldots,r\}$ be such that $\hat{P}_{k,o}(st) \notin \hat{P}_{k,o}(L_{N_k})$. Then, since $\hat{P}_{k,o}^{o} : \Sigma_o^\star \to \hat{\Sigma}_{k,o}^\star$, $\hat{P}_{k,o} : \Sigma^\star \to \hat{\Sigma}_{k,o}^\star$ and $\hat{\Sigma}_{k,o} \subseteq \Sigma_o$, we have that $\hat{P}_{k,o}^{o^{-1}}(\hat{P}_{k,o}(st)) \cap \hat{P}_{k,o}^{o^{-1}}(\hat{P}_{k,o}(L_{N_k})) = \emptyset$. Notice that $P_o(st) \in \hat{P}_{k,o}^{o^{-1}}(\hat{P}_{k,o}(st))$, since $P_o : \Sigma^\star \to \Sigma_o^\star$, and $\Sigma_o = \cup_{k=1}^{r} \hat{\Sigma}_{k,o}$. Therefore, since $\hat{P}_{k,o}^{o^{-1}}(\hat{P}_{k,o}(st)) \cap \hat{P}_{k,o}^{o^{-1}}(\hat{P}_{k,o}(L_{N_k})) = \emptyset$, then $P_o(st) \notin \hat{P}_{k,o}^{o^{-1}}(\hat{P}_{k,o}(L_{N_k}))$. Thus, $P_o(st) \notin \cap_{k=1}^{r} \hat{P}_{k,o}^{o^{-1}}(\hat{P}_{k,o}(L_{N_k})) = \hat{L}_{N_a}$.

($\Leftarrow$) Let us consider that $P_o(st) \notin \hat{L}_{N_a}$. Thus, $\exists k \in \{1,\ldots,r\}$ such that $P_o(st) \notin \hat{P}_{k,o}^{o^{-1}}(\hat{P}_{k,o}(L_{N_k}))$. Notice that any trace $v \in \hat{P}_{k,o}^{o^{-1}}(\hat{P}_{k,o}(L_{N_k}))$ can be written as $v = t_1\sigma_{o_1}t_2\sigma_{o_2}\ldots\sigma_{o_n}t_{n+1}$, where $t_i \in (\Sigma_o \setminus \hat{\Sigma}_{k,o})^\star$, $\sigma_{o_j} \in \hat{\Sigma}_{k,o}$, and $\sigma_{o_1}\sigma_{o_2}\ldots\sigma_{o_n} \in \hat{P}_{k,o}(L_{N_k})$. Let us suppose that $\hat{P}_{k,o}(st) = \hat{P}_{k,o}(v) = \sigma_{o_1}\sigma_{o_2}\ldots\sigma_{o_n}$. Then, according to the definitions of $\hat{P}_{k,o}^{o}$, $\hat{P}_{k,o}$ and P_o, we have that $P_o(st) \in \hat{P}_{k,o}^{o^{-1}}(\hat{P}_{k,o}(L_{N_k}))$, which contradicts the initial assumption that $P_o(st) \notin \hat{P}_{k,o}^{o^{-1}}(\hat{P}_{k,o}(L_{N_k}))$, and therefore, $\hat{P}_{k,o}(st) \notin \hat{P}_{k,o}(L_{N_k})$. ■

Theorem 4.1 *If L is synchronously codiagnosable with respect to L_{N_k}, $\hat{P}_{k,o} : \Sigma^\star \to \hat{\Sigma}_{k,o}^\star$, and Σ_f, then L is synchronously diagnosable with respect to L_{N_k}, $P_{k,o}^{o} : \Sigma_o^\star \to \Sigma_{k,o}^\star$, $P_{k,o} : \Sigma^\star \to \Sigma_{k,o}^\star$, for $k = 1,\ldots,r$, $P_o : \Sigma^\star \to \Sigma_o^\star$, and Σ_f, where $\Sigma_o = \cup_{k=1}^{r} \Sigma_{k,o} = \cup_{k=1}^{r} \hat{\Sigma}_{k,o}$.*

Proof. From Definition 4.1, in order to a language L be synchronously codiagnosable with respect to L_{N_k}, $\hat{P}_{k,o} : \Sigma^\star \to \hat{\Sigma}_{k,o}^\star$, and Σ_f, there must exist at least one local diagnoser $\mathcal{N}_k$, such that $\hat{P}_{k,o}(st) \notin \hat{P}_{k,o}(L_{N_k})$ for any failure trace st with arbitrary long length after the occurrence of the failure event. According to Lemma 4.1, $\exists k \in \{1,\ldots,r\}$ such that $\hat{P}_{k,o}(st) \notin \hat{P}_{k,o}(L_{N_k})$, if, and only if, $P_o(st) \notin \hat{L}_{N_a}$. Since $\hat{P}_{k,o} : \Sigma^\star \to \hat{\Sigma}_{k,o}^\star$, $\hat{P}_{k,o}^{o} : \Sigma_o^\star \to \hat{\Sigma}_{k,o}^\star$, $P_{k,o} : \Sigma^\star \to \Sigma_{k,o}^\star$, $P_{k,o}^{o} : \Sigma_o^\star \to \Sigma_{k,o}^\star$, and $\hat{\Sigma}_{k,o} \subseteq \Sigma_{k,o}$, for $k = 1,\ldots,r$, $P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k})) \subseteq \hat{P}_{k,o}^{o^{-1}}(\hat{P}_{k,o}(L_{N_k}))$. Therefore,

$$L_{N_a} \subseteq \hat{L}_{N_a}. \tag{4.2}$$

According to Equation (4.1), if L is synchronously codiagnosable with respect to L_{N_k}, $\hat{P}_{k,o}$, and Σ_f, then, for all failure traces st with arbitrarily long length after the occurrence of σ_f, $P_o(st) \notin \hat{L}_{N_a}$. Since, according to Equation (4.2), $L_{N_a} \subseteq \hat{L}_{N_a}$, then $P_o(st) \notin L_{N_a}$, which implies, according to Definition 3.1, that the language L is synchronously diagnosable with respect to L_{N_k}, $P_{k,o} : \Sigma^\star \to \Sigma_{k,o}^\star$, $P_{k,o}^{o} : \Sigma_o^\star \to \Sigma_{k,o}^\star$, for $k = 1,\ldots,r$, $P_o : \Sigma^\star \to \Sigma_o^\star$, and Σ_f. ■

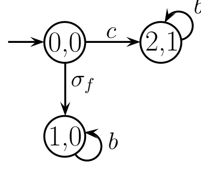


Figure 4.2: Automaton G of Example 4.1.

Remark 4.1 *Notice that, according to Lemma 4.1, to check if* $(\exists k \in \{1,\ldots,r\})$ *such that* $\hat{P}_{k,o}(st) \notin \hat{P}_{k,o}(L_{N_k})$ *for an arbitrarily long failure trace st is equivalent to check if* $P_o(st) \notin \cap_{k=1}^{r} \hat{P}_{k,o}^{o^{-1}}(\hat{P}_{k,o}(L_{N_k}))$ *for* $\Sigma_o = \cup_{k=1}^{r} \hat{\Sigma}_{k,o}$*. Thus, in Definition 3.1 of synchronous diagnosability, the condition* $P_o(st) \notin \cap_{k=1}^{r} P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))$ *can also be replaced by*

$$(\exists k \in \{1,2,\ldots,r\})(P_{k,o}(st) \notin P_{k,o}(L_{N_k})),$$

which shows that the difference between Definition 3.1 of synchronous diagnosability and Definition 4.1 of synchronous codiagnosability lies on the local observable event sets $\Sigma_{k,o}$ *and* $\hat{\Sigma}_{k,o}$*, as pointed out in Corollary 4.1.*

In the following example, we show that the converse of Theorem 4.1 is not always true, *i.e.*, L can be synchronously diagnosable and not synchronously codiagnosable.

Example 4.1 *Consider the system* $G = G_1 \| G_2$ *shown in Figure 4.2, where* G_1 *and* G_2 *are depicted in Figure 4.3. Let us consider the synchronous decentralized diagnosis scheme, where the set of observable events of* G_1 *and* G_2 *are* $\hat{\Sigma}_{1,o} = \{b\}$ *and* $\hat{\Sigma}_{2,o} = \{c\}$*, respectively. In order to do so, consider automata* G_{N_1} *and* G_{N_2}*, depicted in Figure 4.4(a) and 4.4(b), respectively, computed by following Algorithm 3.1. According to Definition 4.1, L is not synchronously codiagnosable with respect to* L_{N_k}*,* $\hat{P}_{k,o}$*, and* Σ_f *if there exist* $st \in L_F$*, with arbitrarily long length after the occurrence of* σ_f*, and nonfailure traces* $s_{N_1} \in L_{N_1}$ *and* $s_{N_2} \in L_{N_2}$ *such that* $\hat{P}_{1,o}(st) = \hat{P}_{1,o}(s_{N_1})$ *and* $\hat{P}_{2,o}(st) = \hat{P}_{2,o}(s_{N_2})$*. Notice that if* $st = \sigma_f b^z$*,* $s_{N_1} = cb^z$*, and* $s_{N_2} = \epsilon$*, where* $z \in \mathbb{N}$*, then* $\hat{P}_{1,o}(st) = \hat{P}_{1,o}(s_{N_1}) = b^z$ *and* $\hat{P}_{2,o}(st) = \hat{P}_{2,o}(s_{N_2}) = \epsilon$*, which shows that L is not synchronously codiagnosable.*

Let us consider now the synchronous centralized diagnosis scheme, where all information regarding the observation of events is communicated to the centralized diagnoser. In this scheme, the set of observable events of G_1 *and* G_2 *are, respectively,* $\Sigma_{1,o} = \{b,c\}$ *and* $\Sigma_{2,o} = \{b,c\}$*. Thus,* $P_{1,o}(st) = b^z$ *and, as it can be seen from Figure 4.4(a), there does not exist a nonfailure trace* $s_{N_1} \in L_{N_1}$ *such that* $P_{1,o}(s_{N_1}) = b^z$*. Thus, L is synchronously diagnosable with respect to* L_{N_k}*,* $P_{k,o}^{o} : \Sigma_o^\star \to \Sigma_{k,o}^\star$*,* $P_{k,o} : \Sigma^\star \to \Sigma_{k,o}^\star$*, for* $k = 1,2$*,* $P_o : \Sigma^\star \to \Sigma_o^\star$*, and* Σ_f*.* □

The following corollary presents a condition that ensures that if L is synchronously diagnosable, then L is synchronously codiagnosable.

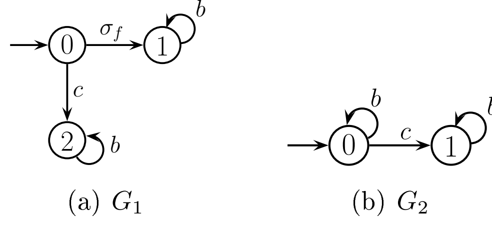


Figure 4.3: Automata G_1 and G_2 of Example 4.1.

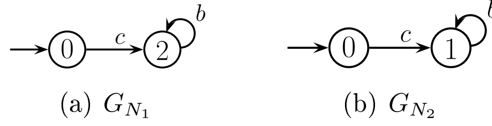


Figure 4.4: Automata G_{N_1} and G_{N_2} of Example 4.1.

Corollary 4.1 *Let $\hat{\Sigma}_{i,uo} \cap \hat{\Sigma}_{j,o} = \emptyset$ for all $i, j \in \{1, \ldots, r\}$. Then, L is synchronously codiagnosable with respect to L_{N_k}, $\hat{P}_{k,o} : \Sigma^\star \to \hat{\Sigma}_{k,o}^\star$, and Σ_f, if, and only if, L is synchronously diagnosable with respect to L_{N_k}, $P_{k,o}^o : \Sigma_o^\star \to \Sigma_{k,o}^\star$, $P_{k,o} : \Sigma^\star \to \Sigma_{k,o}^\star$, for $k = 1, 2$, $P_o : \Sigma^\star \to \Sigma_o^\star$, and Σ_f.*

Proof. Notice that, if $\hat{\Sigma}_{i,uo} \cap \hat{\Sigma}_{j,o} = \emptyset$ for all $i, j \in \{1, \ldots, r\}$, then $\hat{\Sigma}_{k,o} = \Sigma_{k,o}$, and, therefore, $L_{N_a} = \hat{L}_{N_a}$. ■

Remark 4.2 *According to Equations (3.12) and (4.2), we have that $P_o(L_N) \subseteq L_{N_a} \subseteq \hat{L}_{N_a}$. Thus, synchronous codiagnosability implies in synchronous diagnosability, which ultimately implies in the diagnosability of L. The relation between the notions of diagnosability, synchronous diagnosability and synchronous codiagnosability is summarized in Figure 4.5.*

The synchronous codiagnosability verification of the language L can be done by following the steps of Algorithm 3.2 for the verification of synchronous diagnosability,

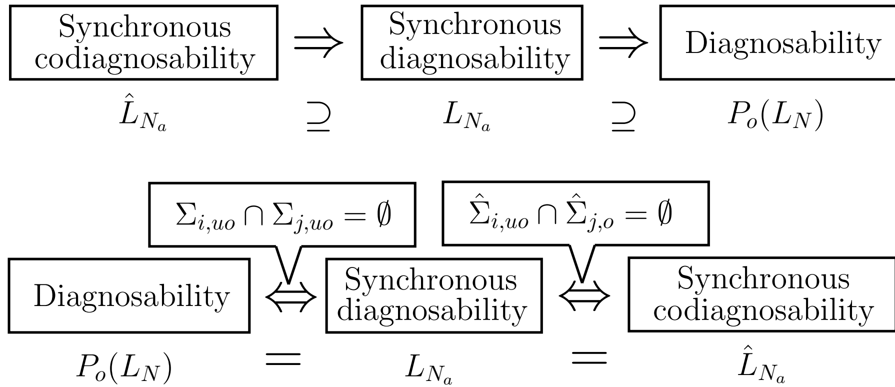


Figure 4.5: Relation between the notions of diagnosability, synchronous diagnosability and synchronous codiagnosability.

replacing, in the definition of the renaming function R_k (Equation (3.13)), in Step 3, the event sets $\Sigma_{k,o}$ and $\Sigma_{k,uo}$, with $\hat{\Sigma}_{k,o}$ and $\hat{\Sigma}_{k,uo}$, respectively, leading to function $\hat{R}_k : \Sigma_{N_k} \rightarrow \hat{\Sigma}^R_{N_k}$, defined as:

$$\hat{R}_k(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \in \hat{\Sigma}_{k,o} \\ \sigma_{R_k}, & \text{if } \sigma \in \hat{\Sigma}_{k,uo} \end{cases} . \tag{4.3}$$

By replacing function R_k (Equation (3.13)) with function $\hat{R}_k$ (Equation (4.3)) in Algorithm 3.2, the synchronous codiagnosability verifier automaton G^{SC}_V is computed and the test for synchronous codiagnosability can be done by searching for cyclic paths in G^{SC}_V formed by states with the label F and not renamed events. In order to prove the correctness of Algorithm 3.2 for the verification of synchronous codiagnosability with function $\hat{R}_k$ (Equation (4.3)), we present the following lemmas.

Lemma 4.2 *Let G_{N_k} be the automaton that models the nonfailure behavior of G_k, and L_{N_k} be the language generated by G_{N_k}, for $k = 1, \ldots, r$. Let $\hat{G}^R_{N_k} = (Q_{N_k}, \hat{\Sigma}^R_{N_k}, \hat{f}^R_{N_k}, q_{0,N_k})$ be the automaton obtained from G_{N_k} by applying function $\hat{R}_{N_k}$ in Step 3 of Algorithm 3.2, and $\hat{L}^R_{N_k}$ be the language generated by $\hat{G}^R_{N_k}$, for $k = 1, \ldots, r$. Then, $\hat{P}^R_o[\cap^r_{k=1}\hat{P}^{R^{-1}}_{N_k}(\hat{L}^R_{N_k})] = \cap^r_{k=1}\hat{P}^R_o[\hat{P}^{R^{-1}}_{N_k}(\hat{L}^R_{N_k})]$, where $\hat{P}^R_o : \hat{\Sigma}^{R\star} \rightarrow \Sigma^\star_o$, $\hat{P}^R_{N_k} : \hat{\Sigma}^{R\star} \rightarrow \Sigma^\star_{N_k}$, for $k = 1, \ldots, r$, and $\hat{\Sigma}^R = \cup^r_{k=1}\hat{\Sigma}^R_{N_k}$.*

Proof. The proof is equal to the proof of Lemma 3.2 if we replace P^R_o, $P^R_{N_k}$, and $L^R_{N_k}$ with $\hat{P}^R_o$, $\hat{P}^R_{N_k}$, and $\hat{L}^R_{N_k}$, respectively. ■

Lemma 4.3 *Let L_{N_k} be the language generated by G_{N_k}. Then,*

$$\cap^r_{k=1}\hat{P}^R_o[\hat{P}^{R^{-1}}_{N_k}(\hat{L}^R_{N_k})] = \cap^r_{k=1}\hat{P}^{o^{-1}}_{k,o}(\hat{P}_{k,o}(L_{N_k})).$$

Proof. The proof is equal to the proof of Lemma 3.3 if we replace P^R_o, $P^R_{N_k}$, $L^R_{N_k}$, $P^o_{k,o}$, and $P_{k,o}$ with $\hat{P}^R_o$, $\hat{P}^R_{N_k}$, $\hat{L}^R_{N_k}$, $\hat{P}^o_{k,o}$, and $\hat{P}_{k,o}$, respectively. ■

We can now state the following theorem that proves the correctness of Algorithm 3.2 for the verification of synchronous codiagnosability with function $\hat{R}_k$ (Equation (4.3)) applied in Step 3 instead of function R_k (Equation (3.13)).

Theorem 4.2 *Let L_{N_k} denote the language generated by G_{N_k}, for $k = 1, \ldots, r$, and $G^{SC}_V = G_F \| \hat{G}^R_N$, where $\hat{G}^R_N = \|^r_{k=1}\hat{G}^R_{N_k}$. Notice that a state of G^{SC}_V is given by $q_V = (q_F, \hat{q}^R_N)$, where q_F and $\hat{q}^R_N$ are states of G_F and $\hat{G}^R_N$, respectively, and $q_F = (q, q_l)$, where $q \in Q$ and $q_l \in \{N, F\}$. Then, L is not synchronously codiagnosable with respect to L_{N_k}, $\hat{P}_{k,o}$, and Σ_f if, and only if, there exists a cyclic path $cl = (q^\delta_V, \sigma_\delta, q^{\delta+1}_V, \ldots, q^\gamma_V, \sigma_\gamma, q^\delta_V)$ in $G^{SC}_V = G_F \| \hat{G}^R_N$, where $\gamma \geq \delta > 0$, such that:*

$$\exists j \in \{\delta, \delta+1, \ldots, \gamma\} \text{ such that for some } q^j_V,$$
$$(q^j_l = F) \wedge (\sigma_j \in \Sigma).$$

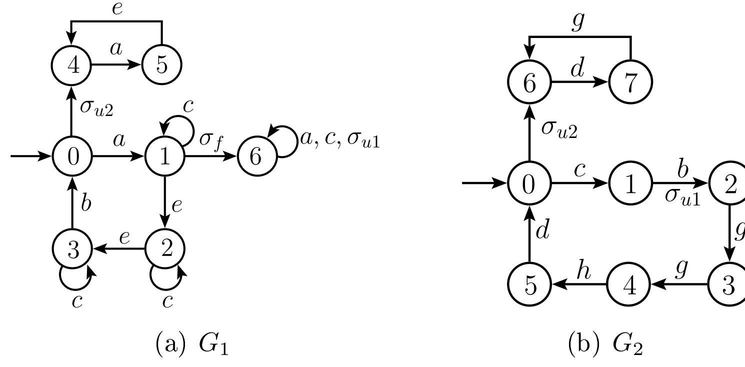


Figure 4.6: Automata G_1 and G_2 of Example 4.2.

Proof. The proof is equal to the proof of Theorem 3.2 if we replace the verifier automaton G_V^{SD} with automaton G_V^{SC}. ■

In the following, we present an example of the synchronous codiagnosability verification of the language of a DES.

Example 4.2 *Consider again the components G_1 and G_2 of the system $G = G_1 \| G_2$, depicted, respectively, in Figures 3.2(a) and 3.2(b). Automata G_1 and G_2 are presented again in Figures 4.6(a) and 4.6(b), respectively. Let $\Sigma = \{a, b, c, d, e, g, h, \sigma_{u1}, \sigma_{u2}, \sigma_f\}$, $\Sigma_o = \{a, b, c, d, e, g, h\}$, $\Sigma_{uo} = \{\sigma_{u1}, \sigma_{u2}, \sigma_f\}$, $\Sigma_f = \{\sigma_f\}$, $\Sigma_1 = \hat{\Sigma}_{1,o} \dot{\cup} \hat{\Sigma}_{1,uo}$, and $\Sigma_2 = \hat{\Sigma}_{2,o} \dot{\cup} \hat{\Sigma}_{2,uo}$, where $\hat{\Sigma}_{1,o} = \{a, b, c, e\}$, $\hat{\Sigma}_{1,uo} = \{\sigma_{u1}, \sigma_{u2}, \sigma_f\}$, $\hat{\Sigma}_{2,o} = \{b, d, g, h\}$, and $\hat{\Sigma}_{2,uo} = \{c, \sigma_{u1}, \sigma_{u2}\}$. In this example, differently from Example 3.2, event c is unobservable to local diagnoser 2, i.e., $c \in \hat{\Sigma}_{2,uo}$. Following Steps 1 and 2 of Algorithm 3.2, automata G_F, G_{N_1}, and G_{N_2} are computed and can be seen in Figures 3.6, 3.5(a) and 3.5(b), respectively. After G_{N_1} and G_{N_2} have been obtained, it is necessary to rename their unobservable events according to Equation (4.3), resulting in automata $\hat{G}_{N_1}^R$ and $\hat{G}_{N_2}^R$ that are depicted in Figures 4.7(a) and 4.7(b), respectively. Automaton $\hat{G}_N^R$ is computed by making the parallel composition between $\hat{G}_{N_1}^R$ and $\hat{G}_{N_2}^R$ in Step 3 of Algorithm 3.2 and it is depicted in Figure 4.8. Since event c is unobservable to local diagnoser 2, then language $\hat{L}_{N_a}$ is a larger set than the language L_{N_a} of Example 3.2 where the synchronous centralized verification is carried out. The growth of the nonfailure language for synchronous decentralized diagnosis can be seen in automaton $\hat{G}_N^R$, and it is represented by all observable transitions related with the gray states, that are states that do not exist in G_N, and the self-loops labeled with event c.*

In Step 4 of Algorithm 3.2, the verifier automaton G_V^{SC} is computed. The states labeled with F of G_V^{SC} and their related transitions can be seen in Figure 4.9. Notice that automaton G_V^{SC} does not have cyclic paths satisfying condition (3.14), and, therefore, language L is synchronously codiagnosable with respect to L_{N_1}, L_{N_2}, $\hat{P}_{1,o}$, $\hat{P}_{2,o}$, and Σ_f.

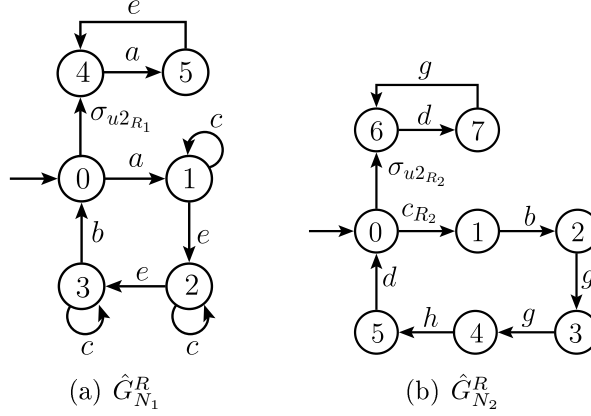


Figure 4.7: Automata $\hat{G}_{N_1}^R$ and $\hat{G}_{N_2}^R$ of Example 4.2.

The delay bound for synchronous decentralized diagnosis can be computed using Algorithm 3.3. In order to do so, automaton G_V^{SC} must be used as input of Algorithm 3.3 instead of automaton G_V^{SD}. In the following example, we compute the delay bound for synchronous decentralized diagnosis for the system G presented in Example 4.2

Example 4.3 *Consider again the system $G = G_1 \| G_2$ presented in Example 4.2. The delay bound for synchronous decentralized diagnosis can be computed by using the verifier automaton G_V^{SC}, where its states labeled with F and their correspondent transitions are depicted in Figure 4.9. By using Algorithm 3.3 with G_V^{SC} as input, the delay bound for synchronous decentralized diagnosis is $z^* = 7$. This result shows that, even with the growth of the nonfailure language for synchronous decentralized diagnosis $\hat{L}_{N_a}$ with respect to language L_{N_a}, the maximum delay bound can be the same for both architectures.*

4.2 Synchronous decentralized failure diagnosis

In order to implement a synchronous decentralized diagnosis scheme, a local diagnoser $\mathcal{N}_k$ must be constructed for each component of the system. In order to do so, it is necessary first to construct the binary state observer Petri net, $\mathcal{N}_{SO_k}$, for each component as described in Algorithm 2.5. Then, a failure detection logic must be attached to each $\mathcal{N}_{SO_k}$, forming the local diagnoser $\mathcal{N}_k$. The failure detection logic indicates the occurrence of the failure event when an event that is not feasible in the current state estimate of one component is observed. Once the occurrence of the failure event is detected, the local diagnoser sends this information to the coordinator that informs the occurrence of the failure event to the operator of the system. According to Lemma 4.1, the failure detection logic proposed in this work is equivalent to verify if the observed trace is in $\hat{L}_{N_a}$. Thus, since $P_o(L_N) \subseteq \hat{L}_{N_a}$, then,

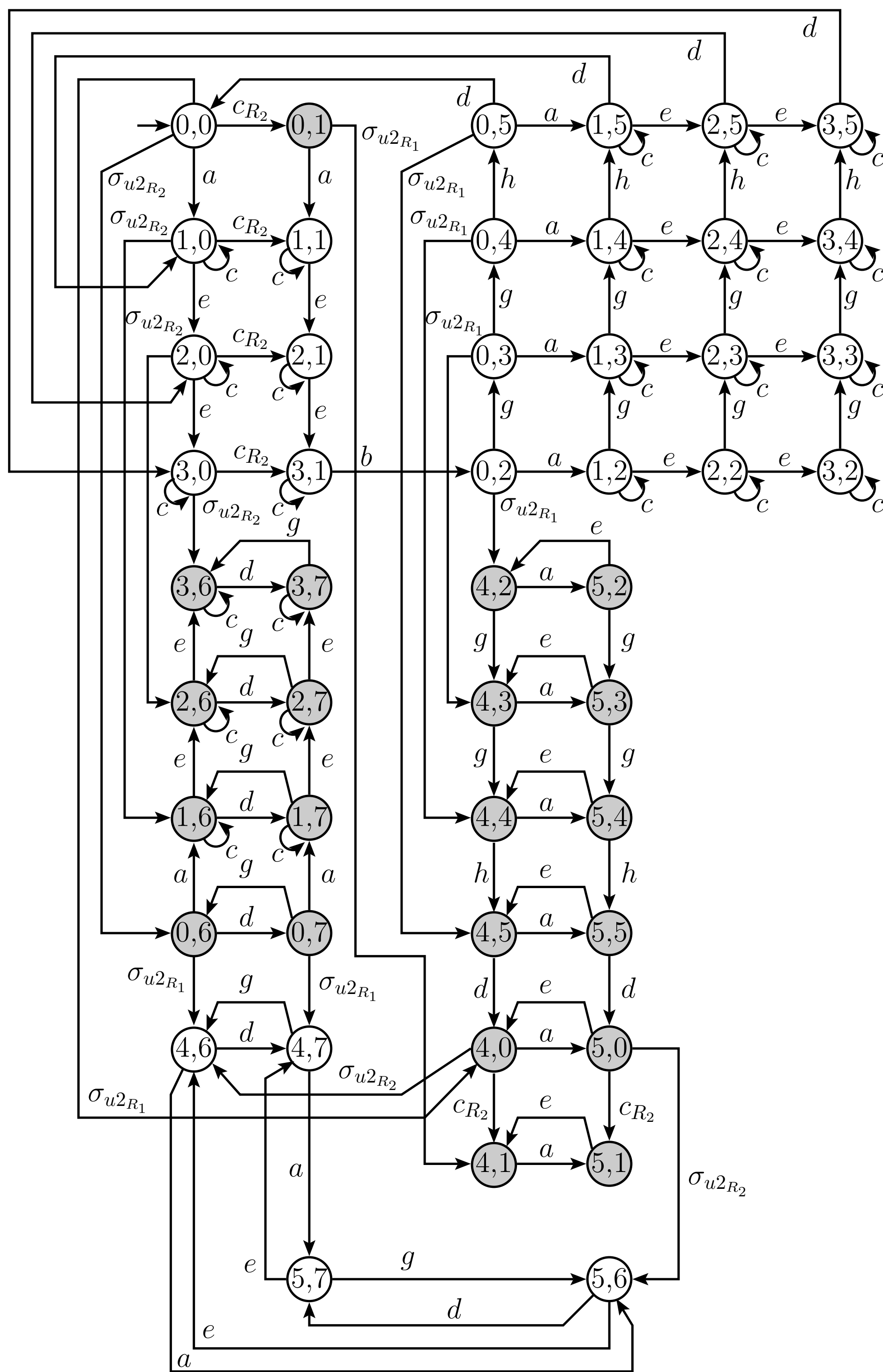


Figure 4.8: Automaton $\hat{G}_N^R$ of Example 4.2.

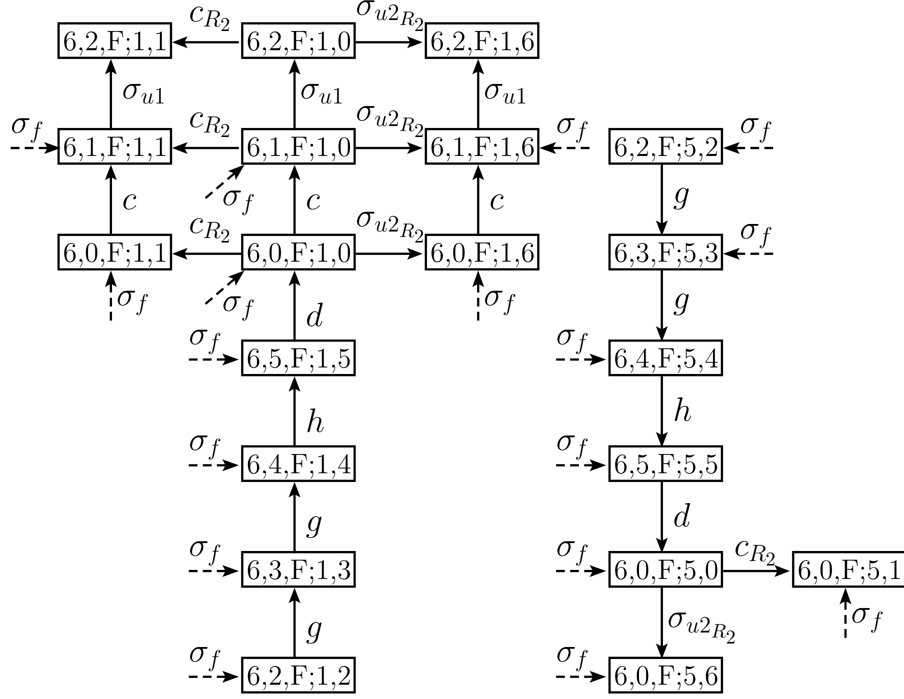


Figure 4.9: Part of automaton G_V^{SC} formed by the states labeled with F and their related transitions of Example 4.2.

if L is synchronously codiagnosable, the proposed diagnoser is capable of detecting the occurrence of a failure trace after a bounded number of observations of events.

Algorithm 4.1 *Local Petri net diagnoser for synchronous decentralized diagnosis.*

Input: *Automata G_{N_k}, for $k = 1, \ldots, r$.*
Output: *Local Petri net diagnosers $\mathcal{N}_k$, for $k = 1, \ldots, r$.*

1: Compute the Petri nets $\mathcal{N}_{D_k} = (P_{D_k}, T_{D_k}, Pre_{D_k}, Post_{D_k}, In_{D_k}, x_{0,D_k}, \Sigma_{o_k}, l_{SO_k})$, for $k = 1, \ldots, r$, from automata G_{N_k} according to Algorithm 3.4.

2: Compute the local Petri net diagnosers $\mathcal{N}_k = (P_k, T_k, Pre_k, Post_k, In_k, x_{0,k}, \Sigma_o, l_{SO_k})$, as follows:

2.1: Add a place p_{F_k} to Petri nets $\mathcal{N}_{D_k}$ and define $Post_k(t_{f_k}, p_F) = 1$, for $k = 1, \ldots, r$. Set $x_{0,}(p_{F_k}) = 0$.

The procedure to perform a decentralized synchronous diagnosis scheme is presented in CABRAL and MOREIRA [83, 84], and the construction of the local diagnosers $\mathcal{N}_k$, for $k = 1, \ldots, r$, is illustrated in the following example.

Example 4.4 *Consider again the system $G = G_1 \| G_2$, where G_1 and G_2 are depicted in Figure 4.6. We present the nonfailure behavior component models*

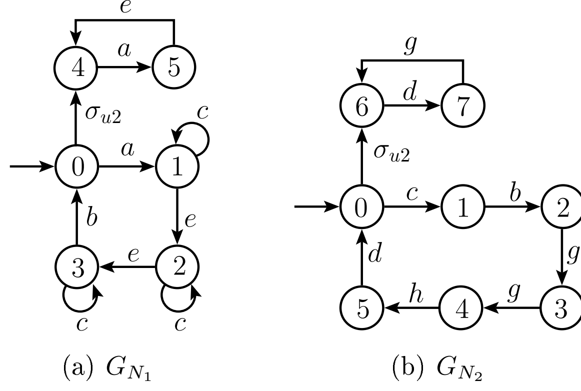
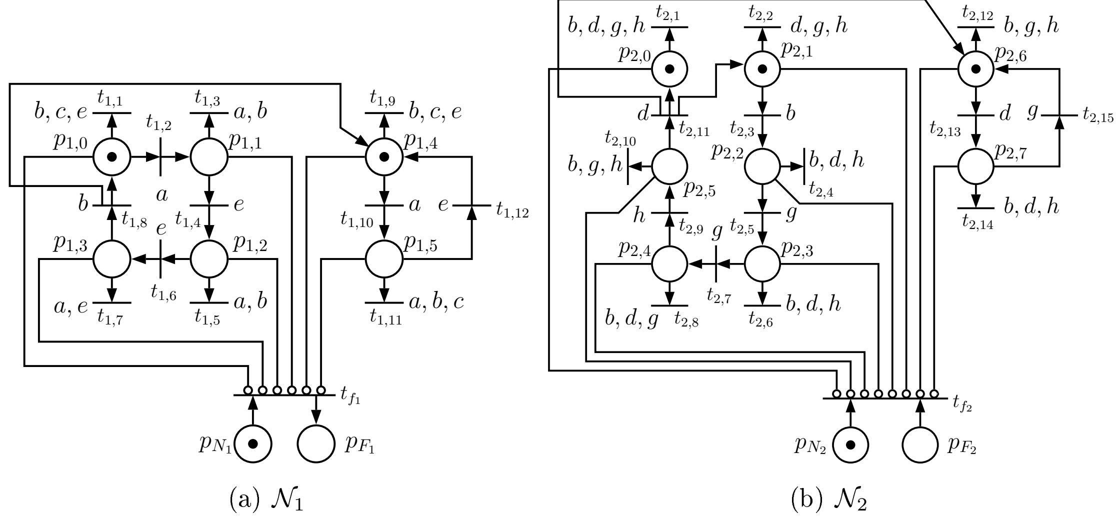


Figure 4.10: Automata G_{N_1} and G_{N_2} of Example 4.4.

G_{N_1} and G_{N_2} in Figure 4.10, where $\hat{\Sigma}_{1,o} = \{a, b, c, e\}$, $\hat{\Sigma}_{1,uo} = \{\sigma_{u1}, \sigma_{u2}, \sigma_f\}$, $\hat{\Sigma}_{2,o} = \{b, d, g, h\}$, and $\hat{\Sigma}_{2,uo} = \{c, \sigma_{u1}, \sigma_{u2}\}$. The local Petri net diagnosers $\mathcal{N}_1$ and $\mathcal{N}_2$ of the modules G_{N_1} and G_{N_2} are computed following Algorithm 4.1 and are shown in Figures 4.11(a) and 4.11(b), respectively. The failure detection logic is represented by transitions t_{f_1} and t_{f_2}, and places p_{N_1}, p_{N_2}, p_{F_1}, and p_{F_2}. Once an observable event that is not feasible in the current state estimate of G_{N_1} or G_{N_2} occurs, transition t_{f_1} or transition t_{f_2} will be enabled and, since it is labeled with the always occurring event, it fires indicating that the failure has been diagnosed.

Let us now consider two failure traces $s_1 = a\sigma_f(a)^z$ and $s_2 = a\sigma_f(c\sigma_{u1}gghd)^z$, for $z \in \mathbb{N}$, in order to illustrate the synchronous decentralized diagnosis. If trace s_1 has been executed by the system, the first module observes the trace $P_{1,o}(s_1) = aa^z$ and the second module observes the trace $P_{2,o}(s_1) = \varepsilon$. When the second occurrence of event a is observed, transitions $t_{1,3}$ and $t_{1,11}$ will fire, removing the tokens from places $p_{1,1}$ and $p_{1,5}$, enabling transition t_{f_1} that fires, removing the token from place p_{N_1} and adding a token to place p_{F_1}, diagnosing the occurrence of the failure event σ_f.

Consider now that the failure trace s_2 has been executed by the system. Trace s_2 is observed by the first module as $P_{1,o}(s_2) = ac$ and by the second module as $P_{2,o}(s_2) = (gghd)^z$. Notice that, if the system executes the failure trace s_2, the local Petri net diagnoser $\mathcal{N}_2$ identifies the occurrence of the failure event, after the observation of event g. When event g is observed, transitions $t_{2,1}$, $t_{2,2}$ and $t_{2,12}$ fire, removing the tokens from places $p_{2,0}$, $p_{2,1}$ and $p_{2,6}$, which enables transition t_{f_2} that fires indicating that the failure has been diagnosed. It is important to notice that, although the local Petri net diagnoser $\mathcal{N}_2$ is constructed based on module G_2 that does not have the failure event modeled, $\mathcal{N}_2$ is necessary to diagnose the failure trace s_2.



(a) $\mathcal{N}_1$ (b) $\mathcal{N}_2$

Figure 4.11: Local Petri net diagnosers $\mathcal{N}_1$ and $\mathcal{N}_2$ of Example 4.4.

4.3 Comparison between modular diagnosability and synchronous codiagnosability

In this section, we compare the notion of synchronous codiagnosability proposed in Definition 4.1 and the notion of modular diagnosability presented in CONTANT *et al.* [59] (Definition 2.25). In order to do so, we consider Assumptions **A1**-**A3** presented in Section 2.4.3 in the definition of synchronous decentralized diagnosability in order to compare the synchronous decentralized diagnosis approach with the modular diagnosis scheme. Let us now rewrite Definition 4.1 according to Lemma 4.1, where it is shown that to check if $\exists k \in \{1, 2 \dots, r\}$ such that $P_{k,o}(st) \notin P_{k,o}(L_{N_k})$ is equivalent to check if $P_o(st) \notin \hat{L}_{N_a}$.

Definition 4.2 (Synchronous codiagnosability) *Let $G_N = \|_{k=1}^{r} G_{N_k}$, and let L_{N_k} denote the language generated by G_{N_k}, for $k = 1, \dots, r$, where r is the number of system components. Then, L is said to be synchronously codiagnosable with respect to $\hat{L}_{N_a}$, P_o, and Σ_f if*

$$(\exists z \in \mathbb{N})(\forall s \in L_F)(\forall st \in L_F, \|t\| \geq z) \Rightarrow P_o(st) \notin \hat{L}_{N_a},$$

where $\hat{L}_{N_a} = \cap_{k=1}^{r} \hat{P}_{k,o}^{o^{-1}}(\hat{P}_{k,o}(L_{N_k}))$.

Now, let us analyze the effects of considering Assumptions **A1**-**A3** in the synchronous codiagnosability definition. According to Assumption **A1**, there are no cycles of unobservable events in the system component models G_k, for $k = 1, \dots, r$. This assumption does not change Definition 4.2 of synchronous codiagnosability of L.

According Assumption **A2**, all common events between two or more modules must be observable. Notice that in the decentralized synchronous diagnosis method, there can be common events between two or more modules that are unobservable to all these modules, and also events that are observable to one component and unobservable to another component. Thus, in order to consider Assumption **A2** in the decentralized synchronous diagnosis approach, we assume that if an event is observable for one module, then it is observable for all modules for which this event is defined, and that there are no common unobservable events between modules. This implies that the synchronization between modules is completely observable, which leads, according to Corollaries 3.1 and 4.1 to the equalities $\hat{L}_{N_a} = L_{N_a} = P_o(L_N)$. Thus, under Assumption **A2**, Definition 4.2 of synchronous codiagnosability becomes equal to Definition 2.22 of diagnosability [14] (and equal to Definition 2.24 of codiagnosability with $\ell = 1$ [17]). It is also important to notice that, as a consequence of Assumption **A2**, a failure event cannot be modeled in more than one component of the system.

As pointed out in Section 2.4.3, in practice, Assumption **A3** excludes traces from L_F that are known to be impossible to be executed by the system, which implies that the failure language of the system can be replaced with language $L_F^{red} \subseteq L_F$. These traces have arbitrarily long length and can be formed with events of all modules, except events from the failure model G_y, for $y \in \{1, \ldots, r\}$. Moreover, as shown in Section 2.4.3, under Assumptions **A1**-**A3**, the condition $\|t\| \geq z$ in Definition 4.2 can be replaced with $\|P_{y,o}(t)\| \geq z'$, where the failure component is G_y, since the diagnosis can be performed only by the local diagnoser associated with component G_y. Therefore, we can conclude that, under assumptions **A1**-**A3**, the definitions of modular diagnosability and synchronous codiagnosability are equal. Hence, modular diagnosability can be seen as a particular case of synchronous decentralized diagnosability. Moreover, if the language L is modularly diagnosable with respect to $\Sigma_o = \cup_{k=1}^{r} \Sigma_{k,o}$ and $\Sigma_f \subseteq \Sigma_y$, then, only local diagnoser $\mathcal{N}_y$ associated with local component G_y must be implemented for diagnosis.

Algorithm 3.2 can be modified in order to check the modular diagnosability of the language L. In order to do so, the failure component model G_y must be identified and the search for cyclic paths performed in Step 5 must be modified in order to take into account only cyclic paths formed by events of G_y. We formalize these modifications in the following algorithm.

Algorithm 4.2 *Modular Diagnosability Verification*

Input: *System modules G_k, where $k = 1, \ldots, r$, failure component model G_y, for $y \in \{1, \ldots, r\}$, and $G = \|_{k=1}^{r} G_k$.*

Output: *Modular diagnosability decision.*

1: Compute automaton G_F that models the failure behavior of G, whose marked language is $L_F = L \setminus L_N$, according to Algorithm 2.7 [71].

2: Compute automaton $G_N = (Q_N, \Sigma_N, f_N, q_0)$ according to Algorithm 2.4 [71].

3: Compute automaton $G_N^R = (Q_N^R, \Sigma^R, f_N^R, q_0)$ as follows:

3.1: Define function $R : \Sigma_N \rightarrow \Sigma_N^R$, as:

$$R(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_o \\ \sigma_R, & \text{if } \sigma \in \Sigma_{uo} \end{cases}. \tag{4.4}$$

3.2: Construct automaton $G_N^R = (Q_N, \Sigma_N^R, f_N^R, q_{0,N})$, with $f_N^R(q_N, R(\sigma)) = f_N(q_N, \sigma)$, $\forall q_N \in Q_N$ and $\forall \sigma \in \Sigma_N$.

4: Compute the verifier automaton $G_V^M = (Q_V, \Sigma_V, f_V, q_{0,V}) = G_F \| G_N^R$. Notice that a state of G_V^M is given by $q_V = (q_F, q_N^R)$, where q_F and q_N^R are states of G_F and G_N^R, respectively, and $q_F = (q, q_l)$, where $q \in Q$ and $q_l \in \{N, F\}$.

5: Verify the existence of a cyclic path $cl = (q_V^\delta, \sigma_\delta, q_V^{\delta+1}, \ldots, q_V^\gamma, \sigma_\gamma, q_V^\delta)$, where $\gamma \geq \delta > 0$, in G_V^M, such that:

$$\exists j \in \{\delta, \delta+1, \ldots, \gamma\} \text{ such that for some } q_V^j,$$
$$(q_l^j = F) \wedge (\sigma_j \in \Sigma_y).$$

If the answer is yes, then L is not modularly diagnosable. Otherwise, L is modularly diagnosable.

Notice that, since in the modular diagnosis approach all unobservable events are private events of the modules of the system, the unobservable event renaming function R of Equation (4.4) can be applied to automaton G_N instead of automata G_{N_k}. In the following theorem, we present the proof of correctness of Algorithm 4.2.

Theorem 4.3 *L is not modularly diagnosable with respect to $\Sigma_o = \cup_{k=1}^r \Sigma_{k,o}$, and $\Sigma_f \subseteq \Sigma_y$ if, and only if, there exists a cyclic path $cl = (q_V^\delta, \sigma_\delta, q_V^{\delta+1}, \ldots, q_V^\gamma, \sigma_\gamma, q_V^\delta)$ in G_V^M, where $\gamma \geq \delta > 0$, such that:*

$$\exists j \in \{\delta, \delta+1, \ldots, \gamma\} \text{ such that for some } q_V^j,$$
$$(q_l^j = F) \wedge (\sigma_j \in \Sigma_y). \tag{4.5}$$

Proof. According to Definition 2.25, in order to verify the modular diagnosability of the language of the system L, it is necessary to check if there exists a failure trace st such that $P_o(st) \notin P_o(L_N)$, and $\|P_{y,o}(t)\| \geq z'$, where $z' \in \mathbb{N}$. In order to check the

modular diagnosability of L, it can be verified if there exists a failure trace st such that $P_{y,o}(st) \in P^R_{y,o}(\mathcal{L}(G^R_N))$, where $P^R_{y,o} : \Sigma^R_N \to \Sigma_{y,o}$. Since the unobservable events of G^R_N are renamed, and hence, are private events of G^R_N, it can be seen that the verifier automaton G^M_V proposed here is equal to the verifier automaton G_V obtained by applying the method proposed in MOREIRA *et al.* [71] to a system whose failure automaton G_F marks L_F and the nonfailure behavior is modeled by G^R_N. Considering Assumption **A3**, the system cannot generate arbitrarily long subtraces formed with events that do not belong to the failure automaton G_y. Thus, the search for cyclic paths in G^M_V must be carried out only for cyclic paths that have events of G_y, which concludes the proof. ■

Remark 4.3 *Recently, an incremental method for the verification of modular diagnosability of DESs has been proposed [63]. The method proposed in LI* et al. *[63] consists in the construction of a local verifier automaton G_{V_y} for the failure component G_y. If the failure component is non-diagnosable, a parallel composition is carried out between verifier G_{V_y} and other components of the system that have common events with G_y. In the worst case, the verifier G_{V_y} must be composed with all*

$$\binom{r-1}{k}, k = 1, 2, \ldots, r-1,$$

possible combination of the remaining modules of the system. Notice that in the method presented in this work for the verification of modular diagnosability, only one verifier G^M_V must be computed. Since in the method proposed in LI et al. *[63] several automata must be constructed to verify modular diagnosability, and there is no way to define previously which is the number of states and transitions of the automata that must be computed, it is impossible to know which one of the methods is the best one in terms of computational cost for modular diagnosability verification, i.e., the best method for the verification of modular diagnosability in terms of computational cost depends on each case.*

In the following example, we illustrate the verification of the modular diagnosability and the implementation of a local Petri net diagnoser built from Algorithm 4.1 for the the system presented in Example 2.13.

Example 4.5 *Consider again the system $G = G_1 \| G_2 \| G_3$ presented in Example 2.13, where G_1, G_2 and G_3 are depicted in Figure 4.12, and automaton G is shown in Figure 4.13. The set of events of G_1, G_2 and G_3 are $\Sigma_1 = \Sigma_{1,uo} \dot\cup \Sigma_{1,o} = \{a, b, \sigma_f\}$, $\Sigma_2 = \Sigma_{2,o} = \{a, c, d, e\}$, and $\Sigma_3 = \Sigma_{3,o} = \{a, c, d, e\}$, respectively, where $\Sigma_{1,uo} = \Sigma_f = \{\sigma_f\}$, $\Sigma_{1,o} = \{a, b\}$, $\Sigma_o = \Sigma_{1,o} \cup \Sigma_{2,o} \cup \Sigma_{3,o} = \{a, b, c, d, e\}$, and $\Sigma_{uo} = \{\sigma_f\}$. The first step to verify the modular diagnosability according to Algorithm 4.2 is to build automaton G_F, depicted in Figure 4.14. In Steps 2 and 3, automaton*

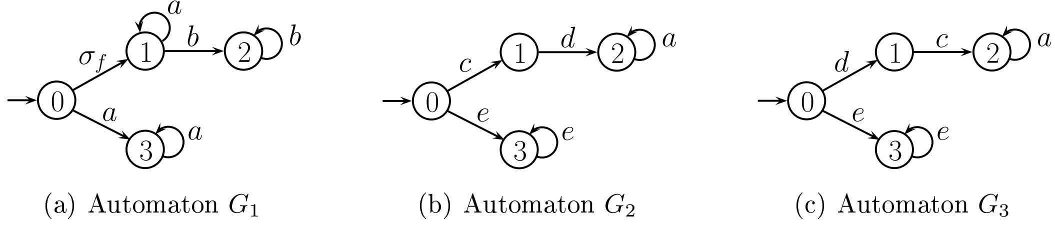


Figure 4.12: Automata G_1, G_2 and G_3 of Example 4.5.

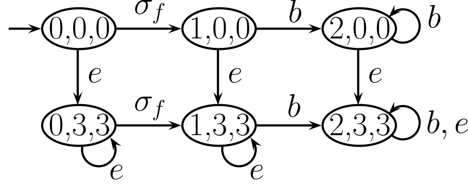


Figure 4.13: Automaton G of Example 4.5.

*G_N^R is obtained by renaming the unobservable events of G_N. We show automaton G_N^R in Figure 4.15. Notice that, in this example, the state transition diagram of G_N and G_N^R are equal. In Step 4 of Algorithm 4.2, the verifier automaton G_V^M is computed by making the parallel composition between automata G_F and G_N^R. Automaton G_V^M is depicted in Figure 4.16. Notice that there exists the cyclic path $((1,3,3,F;0,3,3,N), e, (1,3,3,F;0,3,3,N))$ in G_V^M with an event that does not belong to automaton G_1, which is the failure component model of the system, i.e., $e \notin \Sigma_1$. Thus, the language generated by G, L, is modularly diagnosable with respect to $\Sigma_o = \cup_{k=1}^{r} \Sigma_{k,o}$, and $\Sigma_f \subseteq \Sigma_y$. The fact that the system is modularly diagnosable is guaranteed by Assumption **A3** since, although the system could generate the trace $\sigma_f e^z$, $z \in \mathbb{N}$, for an arbitrarily large value of z, it would contradict Assumption **A3**. Thus, module G_1 eventually will generate event b and the failure event would be diagnosed.*

Let us now illustrate how to perform the diagnosis of the failure event σ_f by using the local Petri net diagnoser $\mathcal{N}_1$. By following Algorithm 4.1, the local Petri net diagnoser $\mathcal{N}_1$, depicted in Figure 4.17, is computed. If the system generates the failure trace $\sigma_f e^z b$, where $z \in \mathbb{N}$, transition $t_{1,1}$ will fire when event b is observed, removing the token from place $p_{1,0}$, which enables transition t_{f_1}. Transition t_{f_1} fires, removing the token from place p_{N_1} and adding a token to place p_{F_1}, diagnosing the

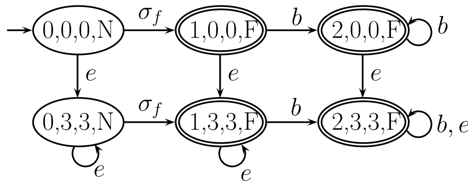


Figure 4.14: Automaton G_F of Example 4.5.

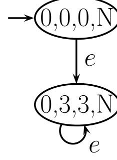


Figure 4.15: Automaton G_N^R of Example 4.5.

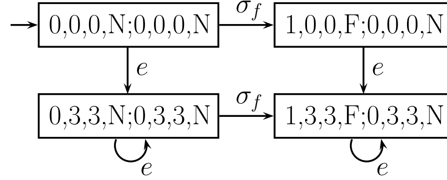


Figure 4.16: Automaton G_V^M of Example 4.5.

occurrence of the failure event σ_f.

4.4 Final remarks

In this chapter, we generalize the notion of synchronous diagnosability to a synchronous decentralized diagnosability. In order to do so, we consider that local diagnosers based on the nonfailure models of the components of the system are implemented locally in a decentralized architecture. The local diagnosers do not communicate among each other and, if a local diagnoser identifies the failure occurrence, it sends this information to a coordinator that indicates the failure occurrence. In this scheme, an event can be observable to a local diagnoser and unobservable to another local diagnoser. The nonfailure language for the synchronous decentralized diagnosis scheme can be a larger set than the nonfailure language for the synchronous centralized diagnosis architecture. Thus, synchronous codiagnosability implies synchronous diagnosability, which, ultimately implies in the diagnosability of the language of the system.

We show that the algorithm for the verification of synchronous diagnosability can be used to verify the synchronous codiagnosability of the language of the system. Moreover, local Petri net diagnosers based on the nonfailure models of the system

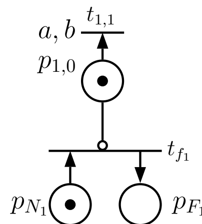


Figure 4.17: Local Petri net diagnoser $\mathcal{N}_1$ of Example 4.5.

components are constructed in order to implement the decentralized synchronous diagnosis scheme. The cube assembly mechatronic system presented in Chapter 3 can also be diagnosed using the decentralized approach proposed in this chapter.

The synchronous decentralized diagnosis scheme presented in this chapter is compared to the modular diagnosis approach presented in CONTANT *et al.* [59]. If we apply all assumptions presented in CONTANT *et al.* [59] to the synchronous decentralized diagnosis scheme, the definition of synchronous codiagnosability becomes equal to the definition of modular diagnosability, which shows that modular diagnosis can be seen as a particular case of synchronous decentralized diagnosis. Moreover, if the system is modularly diagnosable, only the local Petri net diagnoser associated with the failure component need to be computed for failure diagnosis.

In the next chapter, we propose a modification in the local state observer Petri nets in order to decrease the nonfailure language for synchronous diagnosis. This modification consists in adding conditions to the observable transitions of the state observer Petri nets that depend on the marking of the other local Petri nets. These conditions are created based on the global nonfailure behavior model of the system, and, since they can decrease the nonfailure language for synchronous diagnosis, we propose the notion of conditional synchronous diagnosability of DESs.

Chapter 5

Conditional synchronous diagnosability of DESs

In this chapter, we propose a modification in the SPND presented in Chapter 3 with a view to improving the synchronous diagnosis of failure events by reducing the augmented nonfailure language L_{N_a}. The idea is to modify the state observers Petri nets $\mathcal{N}_{SO_k}$, for $k = 1, \ldots, r$, computed by following Algorithm 2.5. Notice that, according to Algorithm 2.5, the set of transitions of $\mathcal{N}_{SO_k}$ is $T_{SO_k} = T_{k,o} \dot{\cup} T'_k$, where $T_{k,o}$ corresponds to all transitions of $\mathcal{N}_{SO_k}$ that are related with observable transitions of G_{N_k}. In order to not allow a transition $t_{k,i} \in T_{k,o}$ to fire if this transition is not associated with a transition in the nonfailure automaton G_N, a condition must be added to the synchronized Petri net diagnoser to avoid the incorrect firing of $t_{k,i}$. Moreover, if the observable event that labels $t_{k,i}$ occurs, and this event is not allowed according to the nonfailure behavior automaton G_N, the token of its input place must be removed, indicating that the state associated with this place does not belong to the current state estimate of the system. In the following example, we illustrate this problem.

Example 5.1 *Consider the system $G = G_1 \| G_2$ presented in Example 3.2, where the state transition diagrams of G_1, G_2 and G are depicted in Figures 5.1(a), 5.1(b), and 5.2, respectively. The event sets of G, G_1 and G_2 are $\Sigma = \{a, b, c, d, e, g, h, \sigma_{u1}, \sigma_{u2}, \sigma_f\}$, $\Sigma_1 = \{a, b, c, e, \sigma_{u1}, \sigma_{u2}, \sigma_f\}$, and $\Sigma_2 = \{b, d, g, h, \sigma_{u1}, \sigma_{u2}\}$, respectively. In Figures 5.3 and 5.4 we show automata G_N and G_N^R, respectively, where G_N^R is computed by following Algorithm 3.2. Notice that the gray states of G_N^R do not exist in automaton G_N and thus, these states and their related observable transitions correspond to the augmented nonfailure language for synchronous diagnosis. Now, consider the synchronized Petri net diagnoser computed for system G in Example 3.4 shown in Figure 5.5.*

Suppose that the system has executed trace $s = s'a = aceeba$, where $s' = aceeb$.

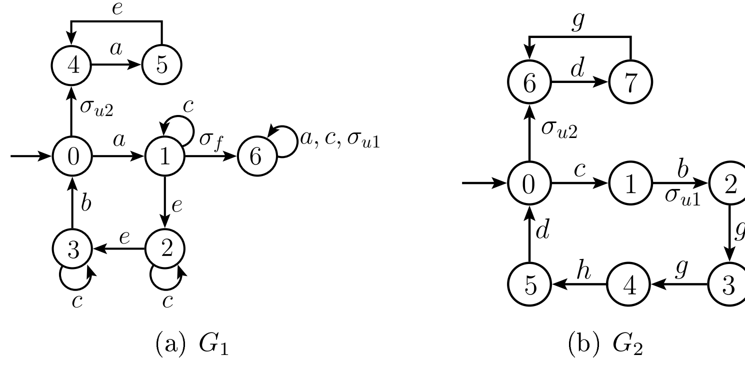


Figure 5.1: Automata G_1 and G_2 of Example 5.1.

After the observation of the prefix s', the places of $\mathcal{N}_D$ that have tokens are $p_{1,0}$, $p_{1,4}$ and $p_{2,2}$, corresponding to the states $(0,2)$ and $(4,2)$ of automaton G_N^R. However, only state $(0,2,N)$ of G_N belongs to the state estimate after the observation of trace s'. After the second observation of event a, transitions $t_{1,2}$ and $t_{1,10}$ fire, removing the tokens from places $p_{1,0}$ and $p_{1,4}$ and adding tokens to places $p_{1,1}$ and $p_{1,5}$. However, state $(5,2,N)$ do not belong to the state estimate of G_N[1] after the observation of trace s, which shows the growth of the nonfailure language for synchronous diagnosis. Since state $(4,2,N)$ does not belong to the state estimate of G_N after the observation of trace s', when the next event a is observed, the token assigned to place $p_{1,4}$ should be removed and no tokens should be added to place $p_{1,5}$, since this place, combined with place $p_{2,2}$, corresponds to a state that does not belong to the state estimate of G_N.

In order to avoid the marking of places that do not correspond to the state estimate of automaton G_N, we can add conditions to the observable transitions of $\mathcal{N}_D$ that allows these transitions to fire only when a corresponding observable transition can occur in G_N. For example, consider transition $t_{1,10}$ of $\mathcal{N}_D$. This transition corresponds to transition $(4,a,5)$ of automaton G_{N_1}, depicted in Figure 5.6(a). Event a can only occur in G_N in states $(0,0,N)$, $(0,2,N)$, $(0,3,N)$, $(0,4,N)$, $(0,5,N)$ $(4,6,N)$ and $(4,7,N)$, where only states $(4,6,N)$ and $(4,7,N)$ have the first coordinate equal to 4. Thus, transition $t_{1,10}$ can fire in $\mathcal{N}_D$ only if place $p_{1,4}$ has a token and place $p_{2,6}$ or place $p_{2,7}$ has a token, since places $p_{1,4}$, $p_{2,6}$ and $p_{2,7}$ correspond to states 4 of G_{N_1}, and states 6 and 7 of G_{N_2}, respectively. This shows that, in this example, the SPND can be modified in order to implement this behavior, decreasing the augmented nonfailure language L_{N_a} for synchronous diagnosis.

[1]Notice that state $(5,2,N)$ does not even exist in the state space diagram of G_N.

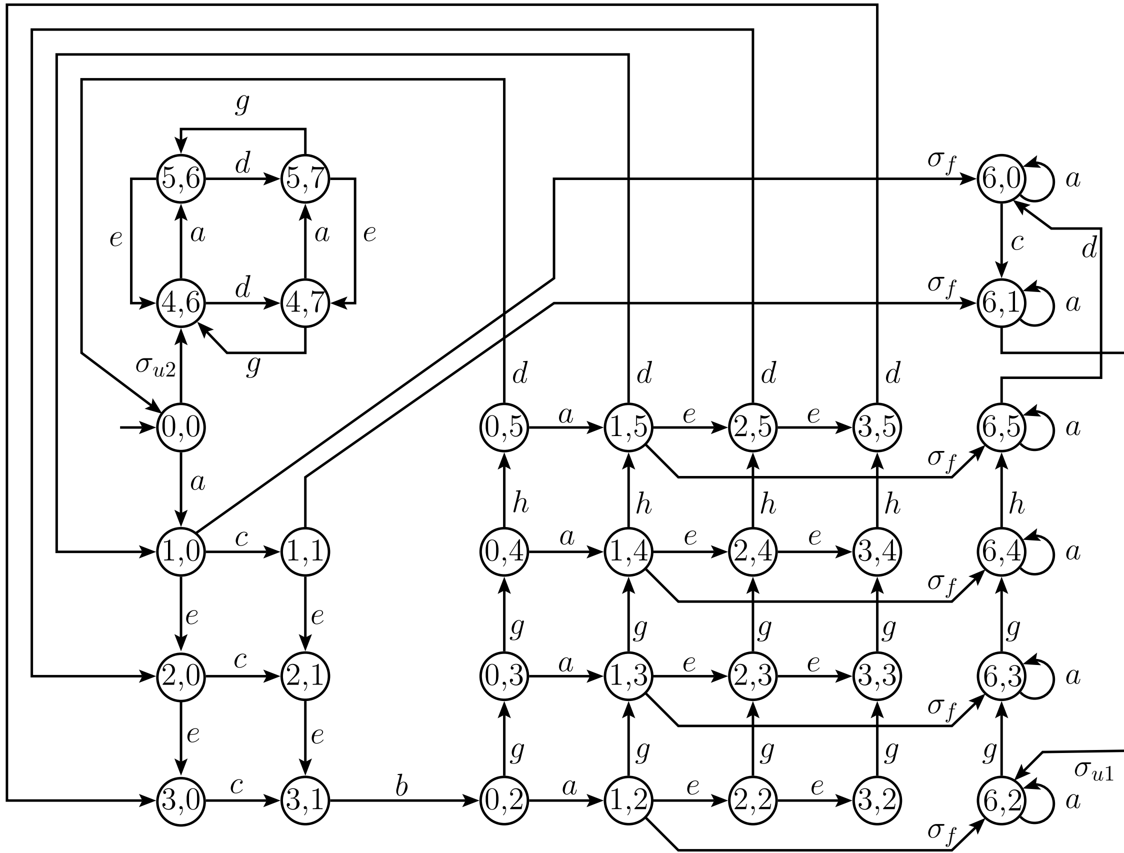


Figure 5.2: Automaton G of Example 5.1.

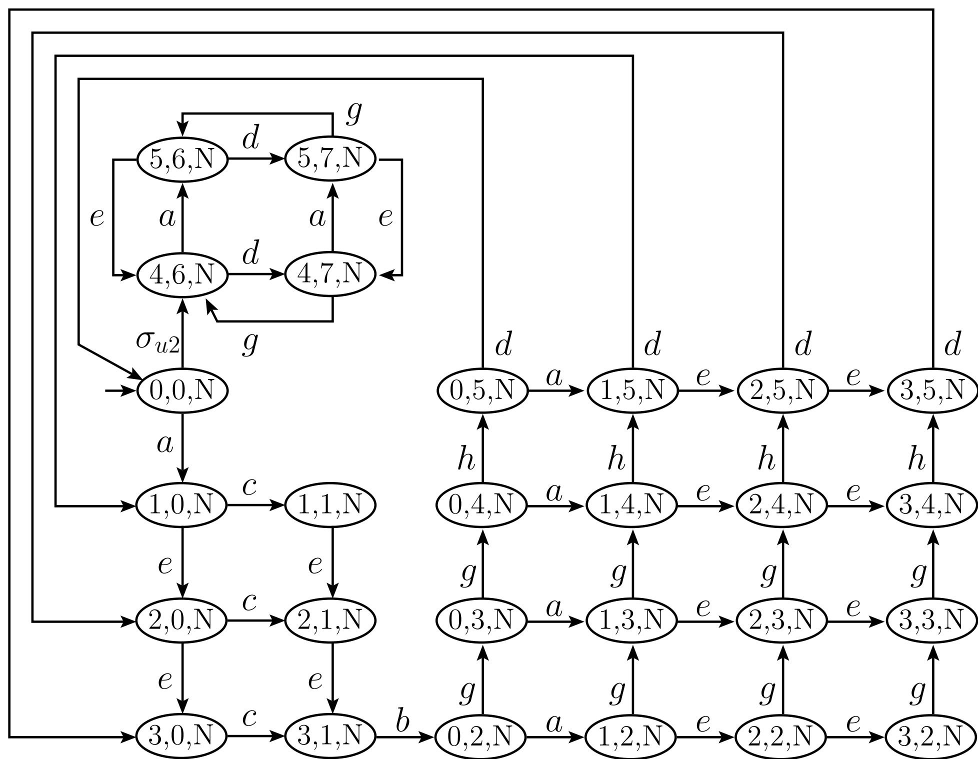


Figure 5.3: Automaton G_N of Example 5.1.

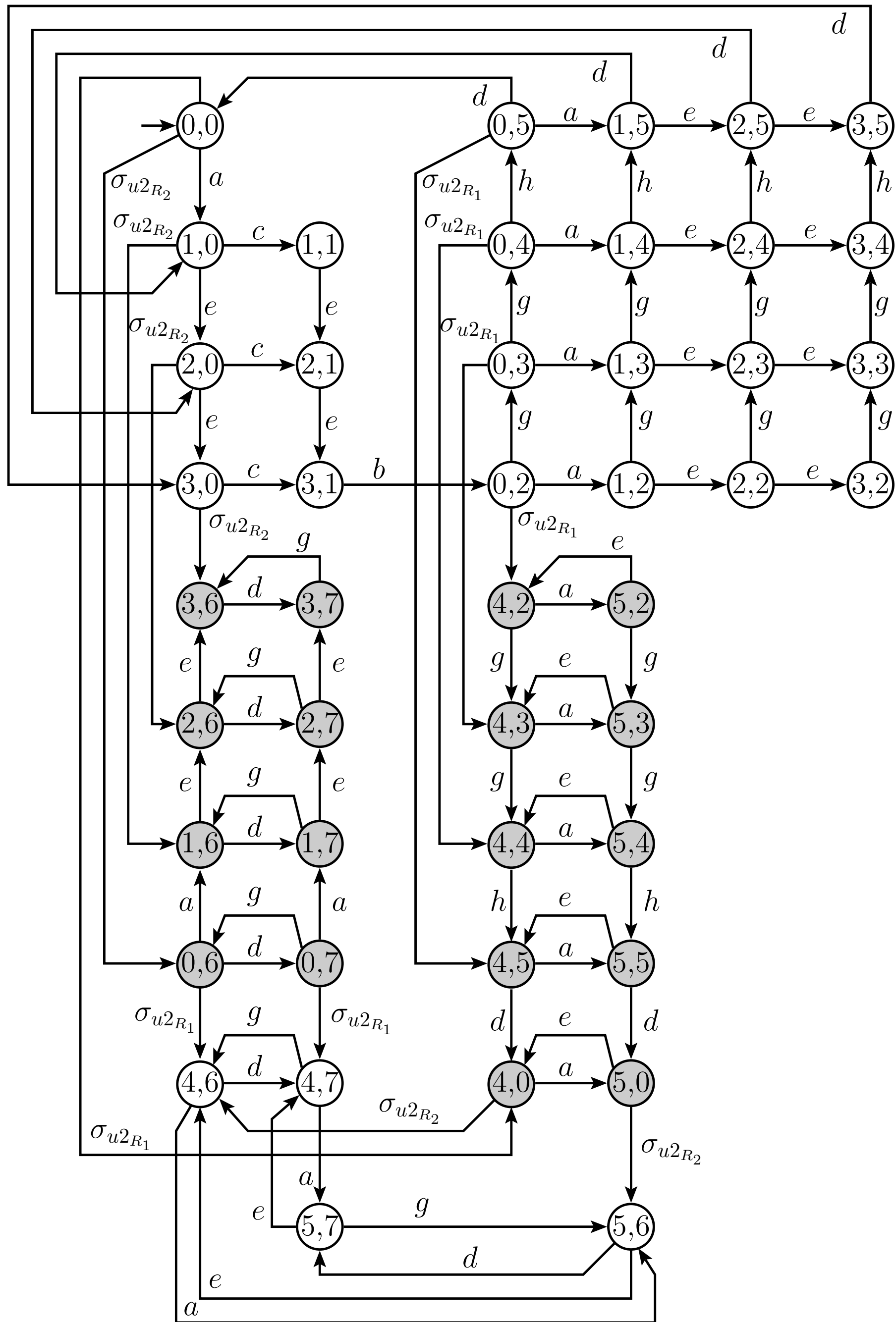


Figure 5.4: Automaton G_N^R of Example 5.1.

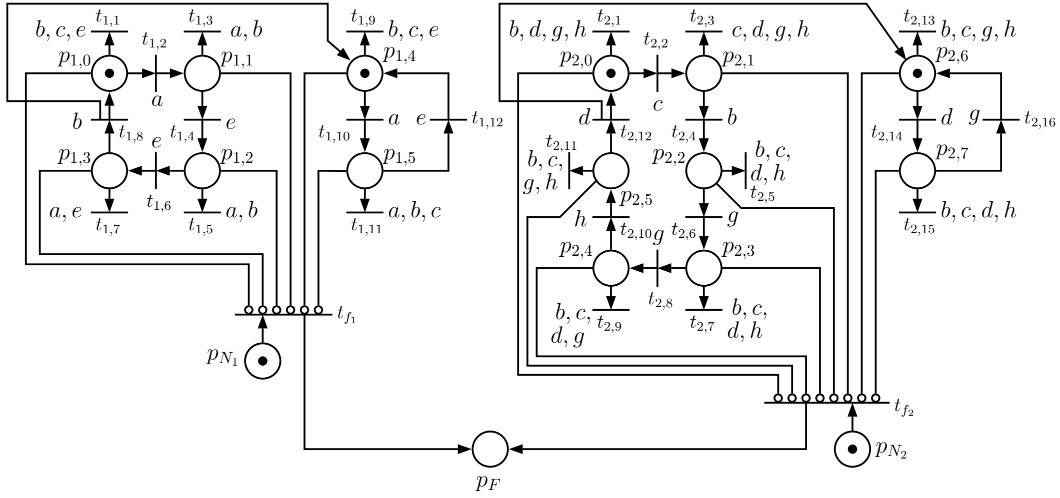


Figure 5.5: Synchronized Petri net diagnoser $\mathcal{N}_D$ of Example 5.1.

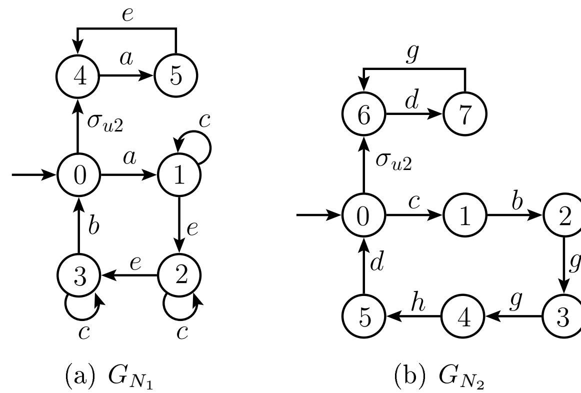


Figure 5.6: Automata G_{N_1} and G_{N_2} of Example 5.1.

5.1 Conditional synchronous Petri net diagnoser

As presented in Example 5.1, the SPND can be modified in order to decrease the augmented nonfailure language for synchronous diagnosis. In order to implement this modification, we add conditions, *i.e.*, boolean expressions, to transitions $t_{k,i} \in T_{k,o}$ of the Petri nets $\mathcal{N}_{SO_k}$, $k = 1, \ldots, r$, that are associated with the marking of the other Petri nets $\mathcal{N}_{SO_j}$, $j = 1, \ldots, r$, and $k \neq j$. If the condition is true, the transition can fire and the state estimate of local $\mathcal{N}_{SO_k}$ is updated, otherwise, we add an output transition with the complementary boolean condition in order to remove the token from the input place of $t_{k,i}$ when the event is observed. This modification leads to an interpreted Petri net called in this work the conditional Petri net state observer $\mathcal{N}^c_{SO_k}$. The conditional Petri net state observer $\mathcal{N}^c_{SO_k}$, $k \in \{1, \ldots, r\}$, is an eight-tuple $\mathcal{N}^c_{SO_k} = (P_{SO_k}, T^c_{SO_k}, Pre^c_{SO_k}, Post^c_{SO_k}, x_{0,SO_k}, \Sigma_{k,o}, C_{SO_k}, l^c_{SO_k})$, where $l^c_{SO_k} : T^c_{SO_k} \to 2^{\Sigma_{k,o}} \times C_{SO_k}$ is a labeling function that associates to each transition in $T^c_{SO_k}$ a set of events from $2^{\Sigma_{k,o}}$ and a condition C from C_{SO_k}, associated with the marking of the places of Petri nets $\mathcal{N}^c_{SO_j}$, for $j = 1, \ldots, r$, $j \neq k$.

In the sequel, we present Algorithm 5.1 for the computation of the conditional Petri net state observers $\mathcal{N}^c_{SO_k}$, for $k = 1, \ldots, r$.

Algorithm 5.1 *Conditional Petri net state observers.*

Input: *Petri net state observers* $\mathcal{N}_{SO_k} = (P_{SO_k}, T_{SO_k}, Pre_{SO_k}, Post_{SO_k}, x_{0,SO_k}, \Sigma_{k,o}, l_{SO_k})$*, for* $k = 1, \ldots, r$*, and automaton* G_N*.*

Output: *Conditional Petri net state observers* $\mathcal{N}^c_{SO_k} = (P_{SO_k}, T^c_{SO_k}, Pre^c_{SO_k}, Post^c_{SO_k}, x_{0,SO_k}, \Sigma_{k,o}, C_{SO_k}, l^c_{SO_k})$*, for* $k = 1, \ldots, r$*.*

1: *Let* $T^{c'}_{SO_k} = \emptyset$*. Create a new transition* t^c_k *for each transition* $\tilde{q}_{N_k} = f_{N_k}(q_{N_k}, \sigma)$ *defined in* G_{N_k}*, where* $\tilde{q}_{N_k}, q_{N_k} \in Q_{N_k}$*, and* $\sigma \in \Sigma_{k,o}$*. For each transition* t^c_k*, define* $Pre^c_{SO_k}(p_k, t^c_k) = 1$*, if* p_k *corresponds to state* q_{N_k}*, and* $Pre^c_{SO_k}(p_k, t^c_k) = 0$*, otherwise, and do* $T^{c'}_{SO_k} = T^{c'}_{SO_k} \cup \{t^c_k\}$*.*

2: *Define* $T^c_{SO_k} = T_{SO_k} \cup T^{c'}_{SO_k}$*.*

3: *Define* $Pre^c_{SO_k} : P_{SO_k} \times T^c_{SO_k} \to \mathbb{N}$ *and* $Post^c_{SO_k} : T^c_{SO_k} \times P_{SO_k} \to \mathbb{N}$ *such that* $Pre^c_{SO_k}(p_k, t_k) = Pre_{SO_k}(p_k, t_k)$*, and* $Post^c_{SO_k}(t_k, p_k) = Post_{SO_k}(t_k, p_k)$ *for all* $p_k \in P_{SO_k}$ *and* $t_k \in T_{SO_k}$*, and* $Post^c_{SO_k}(t^c_k, p_k) = Post^c_{SO_k}(t^c_k, p_k) = 0$*, for all* $t^c_k \in T^{c'}_{SO_k}$ *and* $p_k \in P_{SO_k}$*.*

4: *Define* $l^c_{SO_k} : T^c_{SO_k} \to 2^{\Sigma_{k,o}} \times C_{SO_k}$ *as:*

$$l^c_{SO_k}(t_{k,i}) = \begin{cases} (l_{SO_k}(t_{k,i}), C_{k,i}), & \text{if } t_{k,i} \in T_{k,o} \cup T^{c'}_{SO_k} \\ (l_{SO_k}(t_{k,i}), 1), & \text{otherwise,} \end{cases} \tag{5.1}$$

with

$$C_{k,i} = \begin{cases} [\bigwedge_{j=1, j\neq k}^{r} (\bigvee_{\ell} x_j(p_{j,\ell}))], & \text{if } t_{k,i} \in T_{k,o} \\ \overline{[\bigwedge_{j=1, j\neq k}^{r} (\bigvee_{\ell} x_j(p_{j,\ell}))]}, & \text{if } t_{k,i} \in T_{SO_k}^{c'} \end{cases} \tag{5.2}$$

for all places $p_{j,\ell} \in P_{SO_j}$ such that $I(t_{k,i})$ and $p_{j,\ell}$ correspond to states in Q_{N_k} and Q_{N_j} that are the k-th and j-th coordinates of a state $q_N \in Q_N$, respectively, where $f_N(q_N, \sigma)$ is defined for $\sigma \in l_{SO_k}(t_{k,i})$.

5: Define the initial marking of $\mathcal{N}_{SO_k}^c$ as $x_{0,SO_k}^c = x_{0,SO_k}$, for $k = 1, \ldots, r$.

Notice that, in Algorithm 5.1, the conditions added to the transitions of the Petri net state observers $\mathcal{N}_{SO_k}$, $k = 1, \ldots, r$, in Step 4 depend on the marking of the other Petri net state observers $\mathcal{N}_{SO_j}$, $k = j, \ldots, r$, and $j \neq k$. Since the Petri net state observers are binary Petri nets, we consider that the marking of their places corresponds to a boolean value, *i.e.*, in Equation (5.2), for a given place p, if $x(p) = 0$ its boolean value is equal to false, and if $x(p) = 1$ its boolean value is equal to true. Moreover, these conditions are created based on the global nonfailure behavior of the system. Although the global behavior model of the system G_N need to be computed in order to obtain the conditional state observer Petri nets $\mathcal{N}_{SO_k}^c$, the size of $\mathcal{N}_{SO_k}^c$ are still polynomial in the size of the system component models, avoiding the use of the global plant model for diagnosis. In the following example, we illustrate the construction of the conditional Petri net state observers.

Example 5.2 *Consider the system $G = G_1 \| G_2$, where G_1, G_2 and G are depicted in Figures 5.1(a), 5.1(b), and 5.2, respectively. The event sets of G, G_1 and G_2 are $\Sigma = \{a, b, c, d, e, g, h, \sigma_{u1}, \sigma_{u2}, \sigma_f\}$, $\Sigma_1 = \{a, b, c, e, \sigma_{u1}, \sigma_{u2}, \sigma_f\}$, and $\Sigma_2 = \{b, d, g, h, \sigma_{u1}, \sigma_{u2}\}$, respectively. By using Algorithm 2.5, the Petri net state observers $\mathcal{N}_{SO_1}$ and $\mathcal{N}_{SO_2}$, shown in Figures 5.7(a) and 5.7(b), respectively, are computed from the nonfailure models of the components of the system G_{N_1} and G_{N_2}*

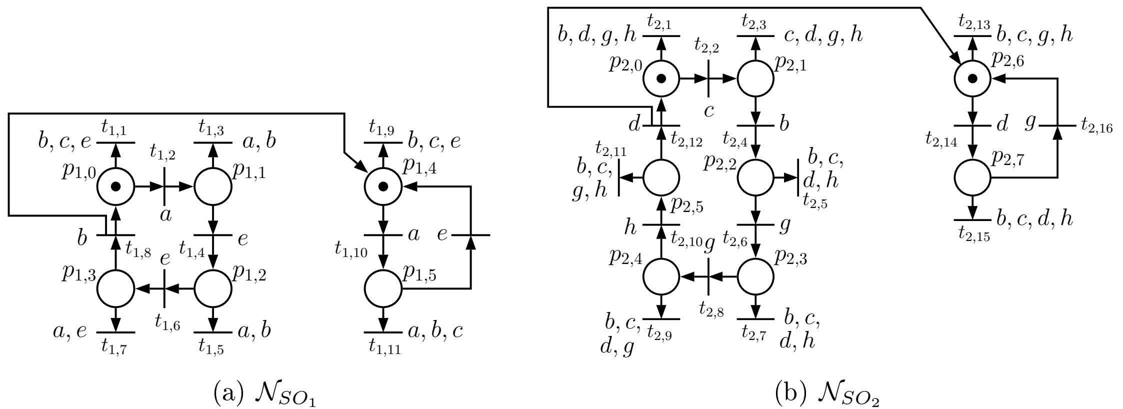


Figure 5.7: State observer Petri nets $\mathcal{N}_{SO_1}$ and $\mathcal{N}_{SO_2}$ of Example 5.3.

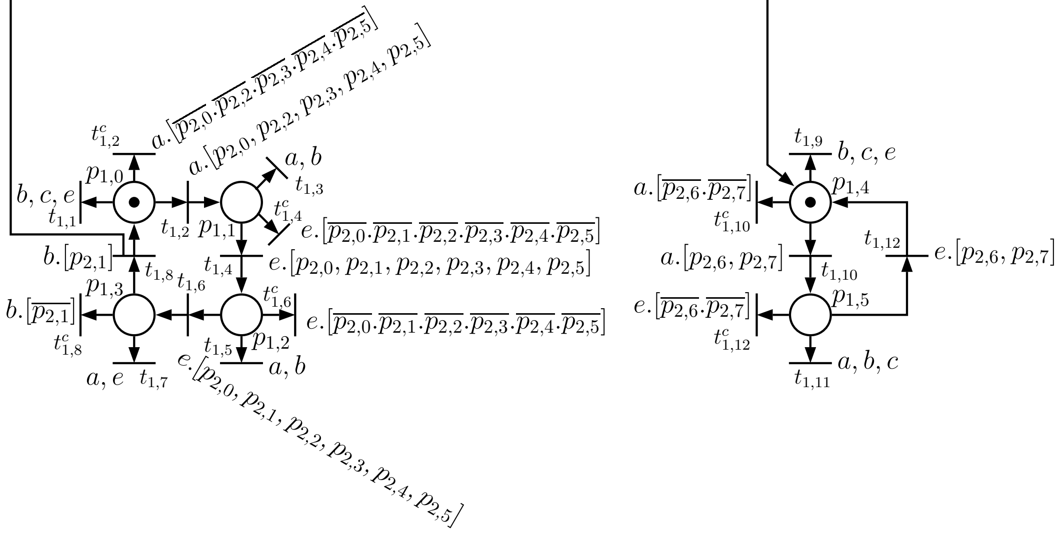


Figure 5.8: Conditional Petri net state observer $\mathcal{N}^c_{SO_1}$ of Example 5.3.

that are illustrated in Figure 5.6. In order to obtain the conditional Petri net state observers $\mathcal{N}^c_{SO_1}$ and $\mathcal{N}^c_{SO_2}$ following Algorithm 5.1, it is necessary to add a transition for each place of $\mathcal{N}^c_{SO_1}$ and $\mathcal{N}^c_{SO_2}$ and associate conditions to all transitions of the Petri net state observer $\mathcal{N}_{SO_1}$ (resp. $\mathcal{N}_{SO_2}$) that depend on the marking of the Petri net $\mathcal{N}_{SO_2}$ (resp. $\mathcal{N}_{SO_1}$). These conditions are obtained from the global nonfailure behavior model of the system G_N, depicted in Figure 5.3. The conditional Petri net state observers $\mathcal{N}^c_{SO_1}$ and $\mathcal{N}^c_{SO_2}$ are shown in Figures 5.8 and 5.9, respectively.

Notice that we have added transitions $t^c_{1,2}$, $t^c_{1,4}$, $t^c_{1,6}$, $t^c_{1,8}$, $t^c_{1,10}$, and $t^c_{1,12}$ to $\mathcal{N}_{SO_1}$ and $t^c_{2,2}$, $t^c_{2,4}$, $t^c_{2,6}$, $t^c_{2,8}$, $t^c_{2,10}$, $t^c_{2,12}$, $t^c_{2,14}$, and $t^c_{2,16}$ to $\mathcal{N}_{SO_2}$. These transitions are added to $\mathcal{N}_{SO_1}$ and $\mathcal{N}_{SO_2}$ in order to compute the conditional Petri net state observers $\mathcal{N}^c_{SO_1}$ and $\mathcal{N}^c_{SO_2}$ in Step 1 of Algorithm 5.1. The conditions are computed and associated to each transition of $\mathcal{N}^c_{SO_1}$ and $\mathcal{N}^c_{SO_2}$ in Step 4 of Algorithm 5.1. All condition predicates associated to the transitions of $\mathcal{N}^c_{SO_1}$ and $\mathcal{N}^c_{SO_2}$ are represented in Figures 5.8 and 5.9 between brackets, where the logical conjunction and logical disjunction are represented by dots and commas, respectively. The marking of the places in these conditions is represented simply by the places. The conditions whose boolean value is always true are not shown in Figures 5.8 and 5.9.

After the conditional state observer Petri nets $\mathcal{N}_{SO_k}$ have been computed, the conditional Petri net diagnoser $\mathcal{N}_{D,c}$ can be obtained following the steps of Algorithm 5.2.

Algorithm 5.2 *Conditional synchronized Petri net diagnoser.*

Input: *Conditional Petri net state observers $\mathcal{N}^c_{SO_k} = (P_{SO_k}, T^c_{SO_k}, Pre^c_{SO_k}, Post^c_{SO_k}, x_{0,SO_k}, \Sigma_{k,o}, C_{SO_k}, l^c_{SO_k})$, for $k = 1, \ldots, r$.*

Output: *Conditional synchronized Petri net diagnoser* $\mathcal{N}_{D,c} = (P^c_D, T^c_D, Pre^c_D, Post^c_D, In^c_D, x^c_{0,D}, \Sigma_o, C^c_D, l^c_D)$, *for* $k = 1, \ldots, r$.

1: *Compute the Petri net* $\mathcal{N}^c_{D_k} = (P^c_{D_k}, T^c_{D_k}, Pre^c_{D_k}, Post^c_{D_k}, In^c_{D_k}, x^c_{0,D_k}, \Sigma_{k,o}, C_{SO_k}, l^c_{SO_k})$, *where* $In^c_{D_k} : P^c_{D_k} \times T^c_{D_k} \rightarrow \{0,1\}$ *denotes the function of inhibitor arcs, as follows:*

1.1: *Add to* $\mathcal{N}^c_{SO_k}$ *a transition* t_{f_k} *labeled with the always occurring event* λ. *Define* $T^c_{D_k} = T^c_{SO_k} \cup \{t_{f_k}\}$.

1.2: *Add to* $\mathcal{N}^c_{SO_k}$ *a place* p_{N_k}, *and define* $Pre^c_{D_k}(p_{N_k}, t_{f_k}) = 1$. *Set* $x^c_{0,D_k}(p_{N_k}) = 1$, *and define* $P^c_{D_k} = P_{SO_k} \cup \{p_{N_k}\}$.

1.3: *Define* $In^c_{D_k}(p^c_{D_k}, t_{f_k}) = 1$ *and* $In_{D_k}(p^c_{D_k}, t^c_{SO_k}) = 0$, $\forall p^c_{D_k} \in P^c_{D_k}$ *and* $\forall t^c_{SO_k} \in T^c_{SO_k}$.

2: *Compute the conditional synchronized Petri net diagnoser* $\mathcal{N}_{D,c} = (P^c_D, T^c_D, Pre^c_D, Post^c_D, In^c_D, x^c_{0,D}, \Sigma_o, C^c_D, l^c_D)$, *as follows:*

2.1: *Form a unique Petri net by grouping all Petri nets* $\mathcal{N}^c_{D_k}$, *for* $k = 1, \ldots, r$.

2.2: *Add a place* p_F *and define* $Post^c_D(t_{f_k}, p_F) = 1$, *for* $k = 1, \ldots, r$. *Set* $x^c_{0,D}(p_F) = 0$.

In the following, we present an example of the computation of the conditional synchronized Petri net diagnoser $\mathcal{N}_{D,c}$ for the system presented in Example 5.1 that illustrates the decrease in the augmented nonfailure language for synchronous diagnosis.

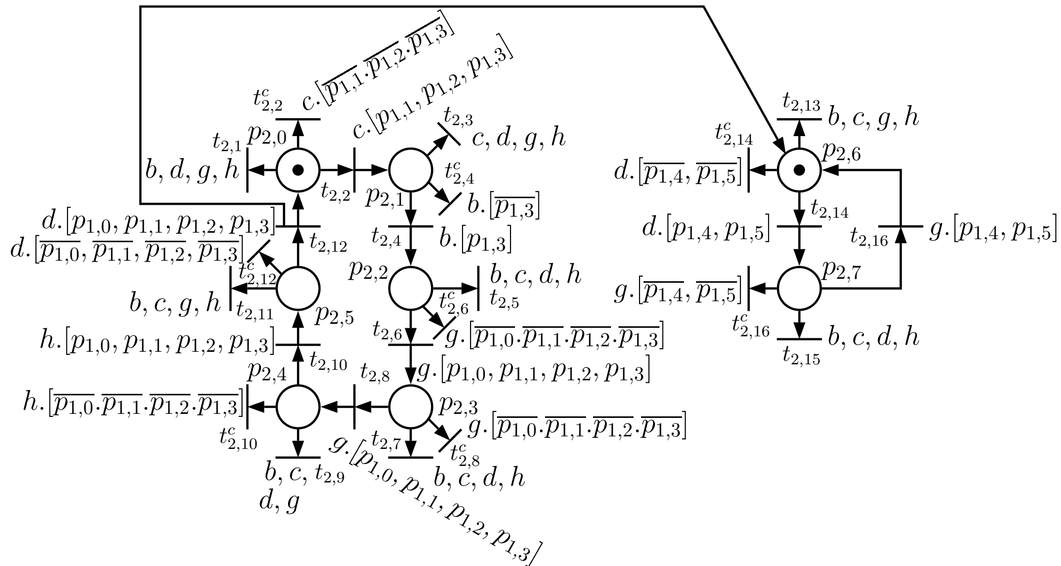


Figure 5.9: Conditional Petri net state observer $\mathcal{N}^c_{SO_2}$ of Example 5.3.

Example 5.3 *In order to compute the conditional synchronized Petri net diagnoser $\mathcal{N}_{D,c}$, according to Algorithm 5.2, it is necessary to group the Petri nets $\mathcal{N}^c_{SO_1}$ and $\mathcal{N}^c_{SO_2}$ and add a failure detection logic to indicate when the failure event has been diagnosed when all tokens of $\mathcal{N}^c_{SO_1}$ or $\mathcal{N}^c_{SO_2}$ have been removed. Due to the lack of space, we do not present the Petri net $\mathcal{N}_{D,c}$ in this work.*

Now, suppose that the system has executed trace $s = s'a = aceeba$, where $s' = aceeb$. After the observation of the prefix s', the places of $\mathcal{N}^c_{SO_1}$ and $\mathcal{N}^c_{SO_2}$ that have tokens are $p_{1,0}$, $p_{1,4}$ and $p_{2,2}$, corresponding to states $(0,2)$ and $(4,2)$ of automaton G^R_N. However, only state $(0,2,N)$ of G_N belongs to the state estimate after the observation of trace s'. After the second observation of event a, transitions $t_{1,2}$ and $t^c_{1,10}$ will fire, removing the tokens from places $p_{1,0}$ and $p_{1,4}$ and adding a token to place $p_{1,1}$. Since the token of place $p_{1,4}$ has been removed as a consequence of the firing of transition $t^c_{1,10}$, differently from Example 5.1, place $p_{1,5}$ will not have tokens after the observation of trace s, indicating that state $(5,2)$ does not belong to the state estimate after the observation of s.

Remark 5.1 *It is important to remark that not all exceeding behavior of the nonfailure language for synchronous diagnosis L_{N_a} with respect to $P_o(L_N)$ is removed with the addition of conditions to the transitions of the Petri net state observers $\mathcal{N}_{SO_k}$ in order to compute the conditional Petri net state observers $\mathcal{N}^c_{SO_k}$.*

In the next section, we introduce the notion of conditional synchronous diagnosability of DESs and present an algorithm to verify this property.

5.2 Conditional synchronous diagnosability

In the previous section, we show how conditions added to the synchronized Petri net diagnoser transitions can decrease the nonfailure language for synchronous diagnosis L_{N_a}. These conditions are added with a view to avoiding the firing of an observable transition, if these transitions are not possible in the nonfailure behavior automaton of the system G_N. This modification leads to the conditional synchronized Petri net diagnoser $\mathcal{N}_{D,c}$. Since the nonfailure language for synchronous diagnosis is decreased, it is necessary to define the notion of conditional synchronous diagnosability of DESs. In order to do so, we first show how to model the augmented nonfailure language for conditional synchronous diagnosis, called $L_{N_{a,c}}$, where $L_{N_{a,c}} \subseteq L_{N_a}$.

In Chapter 3, we show that the augmented nonfailure language for synchronous diagnosis L_{N_a} is equal to the projection in Σ_o of the generated language of automaton G^R_N, *i.e.*, $L_{N_a} = P^R_o(\mathcal{L}(G^R_N))$. Moreover, the conditions added to $\mathcal{N}_{SO_k}$ in order to obtain $\mathcal{N}^c_{SO_k}$ are based on the observable transitions of G_N. Thus, in order to model the nonfailure augmented language for conditional synchronous diagnosis, we

have to erase from G_N^R the observable transitions that do not exist in G_N, leading to automaton $G_{N,c}^R$ whose observable generated language in Σ_o is $P_o^R(\mathcal{L}(G_{N,c}^R)) = L_{N_{a,c}}$. This procedure is formally described in Algorithm 5.3.

Algorithm 5.3 *Conditional nonfailure behavior model*

Input: *Automata G_N and G_N^R.*
Output: *Automaton $G_{N,c}^R$.*

1: *Compute $G_N^{R'}$ by eliminating the transitions $f_N^R(q_N^R, \sigma) = q_N^{R'}$, such that $[(q_N^R \notin Q_N) \vee (q_N^{R'} \notin Q_N)] \wedge (\sigma \in \Sigma_o)$ from G_N^R.*

2: *Compute $G_{N,c}^R = Ac(G_N^{R'})$.*

In the sequel, we present an example to illustrate the computation of automaton $G_{N,c}^R$ according to Algorithm 5.3.

Example 5.4 *Consider automata G_N and G_N^R depicted in Figures 5.3 and 5.4, respectively. Following Algorithm 5.3, automaton $G_{N,c}^R$ is computed by erasing from G_N^R all observable transitions that do not exist in G_N and taking the accessible part of the resulting automaton. Automaton $G_{N,c}^R$ is shown in Figure 5.10. Notice that all observable transitions that reach or depart from the gray states, i.e., states that do not exist in G_N, are erased from G_N^R in order to obtain $G_{N,c}^R$.*

As stated in Remark 5.1, even with the elimination of the observable transitions of G_N^R that do not exist in G_N, the nonfailure language for conditional synchronous diagnosis $L_{N_{a,c}}$ can still be a larger set than the observable nonfailure language of the system $P_o(L_N)$. Thus, even if the language L of a system is diagnosable, L is not necessarily conditionally synchronously diagnosable. This leads to the following definition of conditional synchronous diagnosability.

Definition 5.1 *Let L and $L_N \subset L$ denote the languages generated by G and G_N, respectively, and let $L_F = L \backslash L_N$. Consider that the system is composed of r modules, such that $G_N = \|_{k=1}^r G_{N_k}$, where G_{N_k} is the automaton that models the nonfailure behavior of G_k, and let L_{N_k} denote the language generated by G_{N_k}, for $k = 1, \ldots, r$. Then, L is said to be conditionally synchronously diagnosable with respect to $L_{N_{a,c}}$ and Σ_f if*

$$(\exists n \in \mathbb{N})(\forall s \in L_F)(\forall st \in L_F, \|t\| \geq n) \Rightarrow$$
$$P_o(st) \notin L_{N_{a,c}},$$

where $L_{N_{a,c}} = P_o^R(\mathcal{L}(G_{N,c}^R))$ and $G_{N,c}^R$ is computed according Algorithm 5.3.

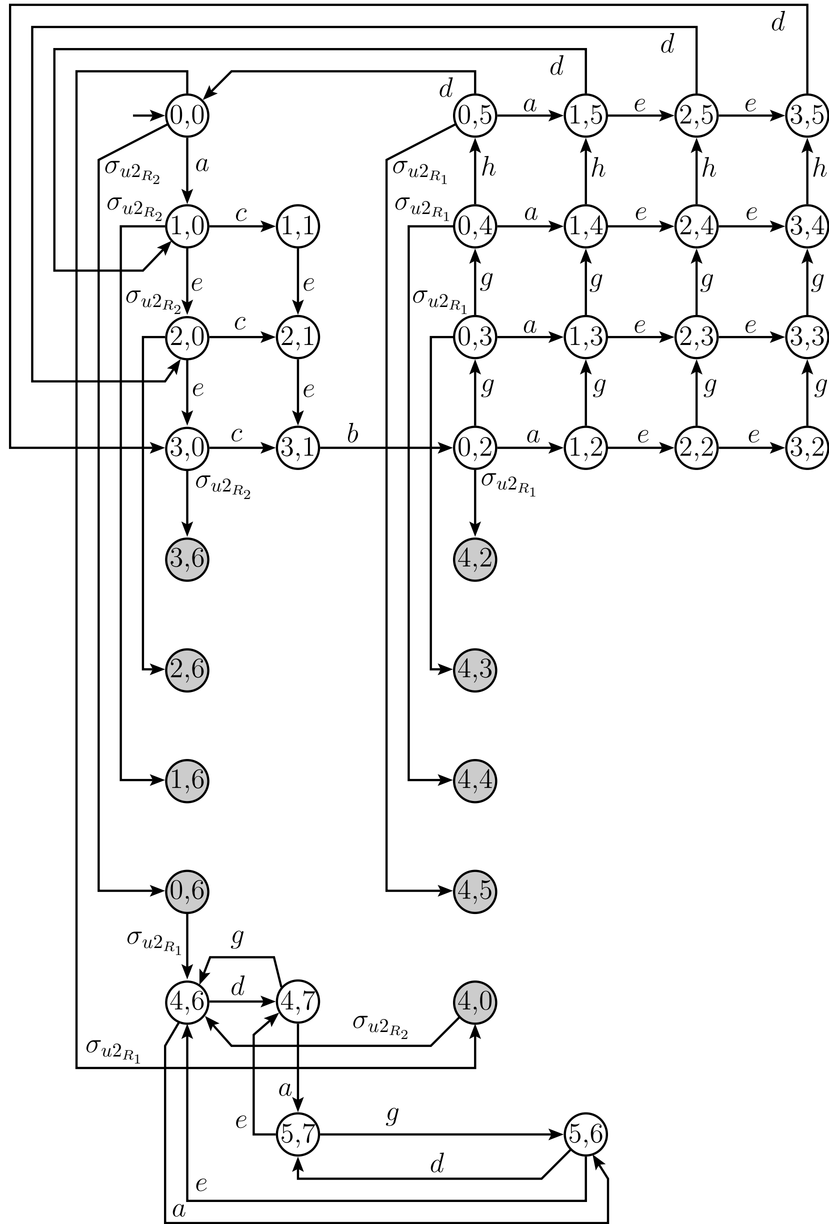


Figure 5.10: Automaton $G^R_{N,c}$ of Example 5.4.

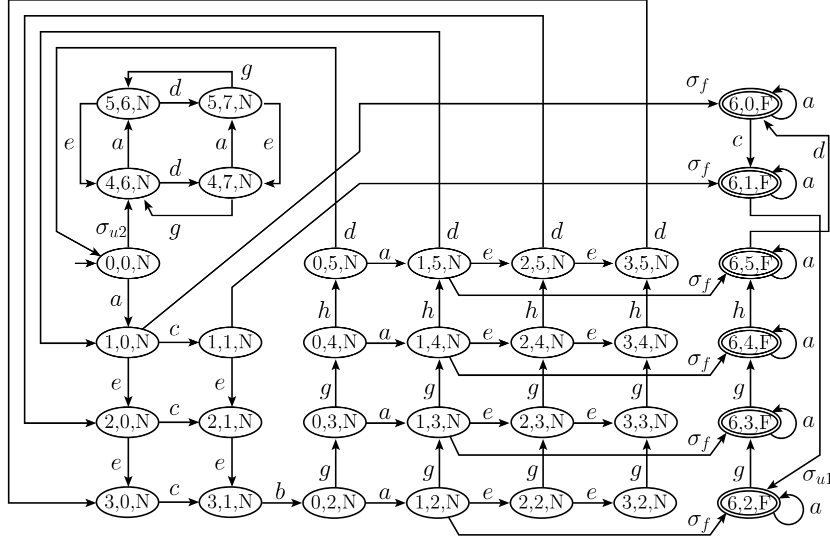


Figure 5.11: Automaton G_F of Example 5.5.

According to Definition 5.1, in order to verify if the language of a system is conditionally synchronously diagnosable, it is necessary to verify if there is an arbitrarily long length failure trace with the same observation as a nonfailure trace that belongs to $L_{N_{a,c}}$. Since all unobservable events of $G^R_{N,c}$ are particular events with respect to automaton G_F, that models the failure language of the system, in order to verify the conditional synchronous diagnosability of a system, Algorithm 3.2 for the verification of synchronous diagnosability can be used. In order to do so, instead of using $G^{SD}_V = G_F \| G^R_N$, it is necessary to build $G^{SD}_{V,c} = G_F \| G^R_{N,c}$ and search for cyclic paths formed with states labeled with F and events that are not renamed. If there exists a cyclic path in $G^{SD}_{V,c}$ with these characteristics, then the system is not conditionally synchronously diagnosable. We illustrate the construction of $G^{SD}_{V,c}$ in the following example.

Example 5.5 *Consider automaton G_F of Example 3.2 depicted in Figure 5.11, and automaton $G^R_{N,c}$ depicted in Figure 5.10. Automaton $G^{SD}_{V,c} = G_F \| G^R_{N,c}$ is shown in Figure 5.12. Notice that there are no cyclic paths in $G^{SD}_{V,c}$ formed with states labeled with F and events that are not renamed. Thus, the language generated by system G, L, is conditionally synchronously diagnosable with respect to $L_{N_{a,c}}$ and Σ_f.*

In order to prove that the conditional synchronized Petri net diagnoser $\mathcal{N}_{D,c}$, obtained from Algorithm 5.3, can be used for synchronous diagnosis, we first introduce the following lemma that states that if a system is synchronous diagnosable, then it is conditionally synchronously diagnosable.

Lemma 5.1 *Let L_F be the language marked by G_F, which models the failure behavior of the system model $G = \|_{k=1}^{r} G_k$, and let $L_{N_{a,c}} = P^R_o(\mathcal{L}(G^R_{N,c}))$, where $G^R_{N,c}$ is*

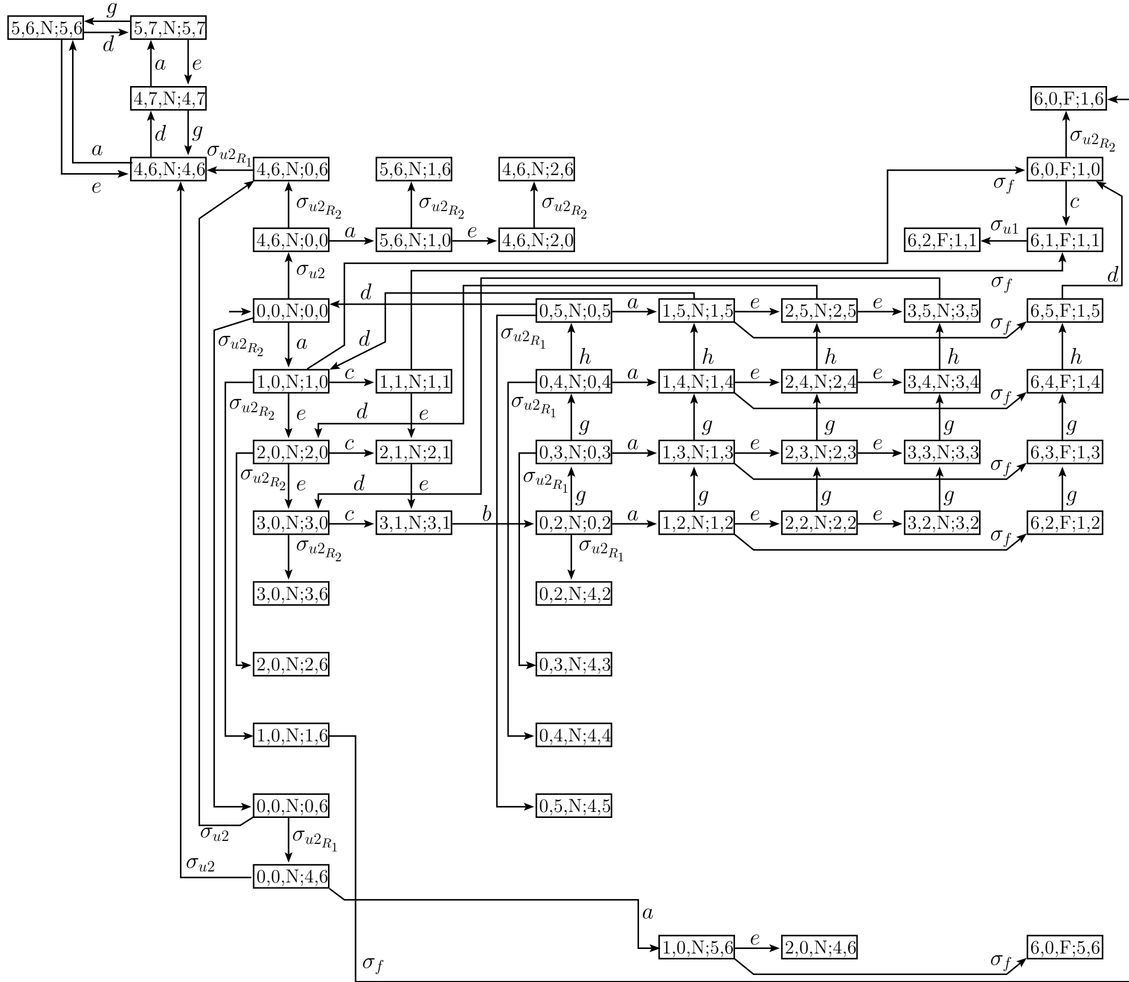


Figure 5.12: Verifier automaton $G_{V,c}^{SD}$ of Example 5.5.

the automaton computed by following Algorithm 5.3. Then, if L is synchronously diagnosable with respect to L_{N_k}, $P_{k,o}^o$, $P_{k,o}$, $k = 1\ldots, r$ and Σ_f, then L is conditionally synchronously diagnosable with respect to $L_{N_{a,c}}$ and Σ_f.

Proof. The proof is straightforward and is based on the construction of automaton $G_{N,c}^R$ according to Algorithm 5.3. Since $G_{N,c}^R$ is obtained by erasing observable transitions of G_N^R that do not exist in G_N and taking the accessible part of the result, $\mathcal{L}(G_{N,c}^R) \subseteq \mathcal{L}(G_N^R)$. Thus, $P_o^R(\mathcal{L}(G_{N,c}^R)) \subseteq P_o^R(\mathcal{L}(G_N^R))$, *i.e.*, $L_{N_{a,c}} \subseteq L_{N_a}$. ■

In the sequel, we present a theorem that ensures that the removal of observable transitions from G_N^R by following the steps of Algorithm 5.3 in order to compute $G_{N,c}^R$ has the same effect as the conditions added to $\mathcal{N}_D$ in order to obtain $\mathcal{N}_{D,c}$. In other words, the conditional synchronized Petri net diagnoser obtained from Algorithm 5.2 can be used for the conditional synchronous diagnosis of DESs.

Theorem 5.1 *Let L_F be the language marked by G_F, which models the failure behavior of the system model $G = \|_{k=1}^r G_k$, and let $L_{N_{a,c}} = P_o^R(\mathcal{L}(G_{N,c}^R))$, where $G_{N,c}^R$ is the automaton computed by following the steps of Algorithm 5.3. Consider language $L_{a,c} = L_{N_{a,c}} \cup \overline{L}_F$, and assume that L is conditionally synchronously diagnosable with respect to $L_{N_{a,c}}$ and Σ_f. Let $s \in L_F$ such that $\forall \omega \in L_{a,c}$ satisfying $P_o(\omega) = P_o(s)$, $\omega \in L_F$. Then, the number of tokens in place p_F of $\mathcal{N}_{D,c}$, after the observation of trace $P_o(s)$, is one.*

Proof. Notice that, in Chapter 3, it is shown that the synchronous Petri net diagnoser $\mathcal{N}_{SO_k}$ provides the state estimate of the nonfailure behavior of the system modules G_{N_k}. Thus, the synchronized Petri net diagnoser $\mathcal{N}_D$ provides the state estimate of automaton G_N^R. In order to compute $G_{N,c}^R$, the observable transitions of G_N^R related to states that do not exist in G_N are erased, according to Algorithm 5.3. Now, consider the conditions associated with the transitions of the Petri nets $\mathcal{N}_{SO_k}^c$ according to Algorithm 5.1, and that the conditional Petri net diagnoser $\mathcal{N}_{D,c}$ is formed by grouping Petri nets $\mathcal{N}_{SO_k}^c$, $k = 1, \ldots, r$ according to Algorithm 5.2. Notice that according to Step 4 of Algorithm 5.1, if an event σ_o, that labels an enabled transition $t_{k,i} \in T_{o_k}$, is observed for a given marking of $\mathcal{N}_{D,c}$, $t_{k,i}$ will fire only if there exists a combination of its input place $p_{k,i}$ with the places of the Petri nets $\mathcal{N}_{SO_j}^c$ that have tokens, for $j = 1, \ldots, r$ and $j \neq k$, corresponding to a state of G_N that also have σ_o active. Otherwise, transition $t_{k,i}^c \in T'_{SO_k}$ will fire, removing the token from $p_{k,i}$, which corresponds to erasing an observable transition from G_N^R that do not exist in automaton G_N. Thus, the Petri net $\mathcal{N}_{D,c}$ provides the state estimate of automaton $G_{N,c}^R$ and, according to Definition 5.1, if the system executed an unambiguous trace s, $P_o(s) \notin L_{N_{a,c}}$. Therefore, if the system executes an unambiguous failure trace, at least one conditional Petri net state observer $\mathcal{N}_{SO_k}^c$,

$k \in \{1, \ldots, r\}$, will lose all its tokens, enabling transition t_{f_k} that fires, adding a token to place p_F. ■

Remark 5.2 *It is important to notice that $L_{N_{a,c}}$ can be a smaller set than L_{N_a}. This fact shows that systems that are not synchronously diagnosable can be conditionally synchronously diagnosable, or the delay bound for conditional synchronous diagnosis can be smaller than the delay bound for synchronous diagnosis.*

A relation between all notions of synchronous diagnosability and synchronous codiagnosability presented in this work can be stated by using the relation between the nonfailure augmented languages $L_{N_{a,c}}$, L_{N_a} and $\hat{L}_{N_a}$. Since $P_o(L_N) \subseteq L_{N_{a,c}} \subseteq L_{N_a} \subseteq \hat{L}_{N_a}$, the synchronous codiagnosablity implies the synchronous diagnosability, that implies the conditional synchronous diagnosability of L, which ultimately implies the diagnosability of L.

In the sequel, we present an example that shows that a system can be not synchronously diagnosable and be conditionally synchronously diagnosable.

Example 5.6 *Consider the system $G = G_1 \| G_2$, where G_1 and G_2 are depicted in Figures 5.13(a), 5.13(b), respectively. The set of events of G_1 and G_2 are $\Sigma_1 = \{a, c, e, g, \sigma_u\}$ and $\Sigma_2 = \{e, h, \sigma_u, \sigma_{u2}, \sigma_f\}$, respectively, where $\Sigma_{1,o} = \{a, c, e, g\}$, $\Sigma_{2,o} = \{e, h\}$, $\Sigma_{1,uo} = \{\sigma_u\}$, $\Sigma_{2,uo} = \{\sigma_u, \sigma_{u2}, \sigma_f\}$, and σ_f is the failure event. Automaton G_F is shown in Figure 5.14. Notice that automata G_F and G are equal, except for the marked states and the labels N and F. We show automaton G_N^R in Figure 5.15. It can be seen that the language L is not synchronously diagnosable with respect to L_{N_1}, L_{N_2}, $P_{1,o}^o$, $P_{2,o}^o$, $P_{1,o}$, $P_{2,o}$, and Σ_f. In order to see this fact, consider that the system has executed the failure trace $s = h\sigma_f eh(eh)^z$, for $z \in \mathbb{N}$. Notice that, in G_N^R, trace $h\sigma_{u_{R_2}} eh(\sigma_{u_{R_2}} eh)^z$ has the same observation in Σ_o, and, consequently, the system G is not synchronously diagnosable.*

Now, consider automaton $G_{N,c}^R$ computed from Algorithm 5.3 shown in Figure 5.16. Notice that there are no traces in $G_{N,c}^R$ with the same observation as trace $s = h\sigma_f eh(eh)^z$. In fact, L is conditionally synchronously diagnosable with respect to $L_{N_{a,c}}$ and Σ_f. This example shows that if we refine the synchronous diagnosis by using more information regarding the nonfailure automaton model G_N, systems that are not synchronously diagnosable can be conditionally synchronously diagnosable.

5.3 Conditional synchronized Petri net diagnoser for an automated system

Consider again the system $G_p = G_{cb} \| G_{hu}$ presented in Section 3.5. The set of events of G_{cb} and G_{hu} are $\Sigma_{cb} = \{a_1, a_2, cr_{on}, c_{off}, a_{pc}, a_{dc}, cl_{on}, d_c\}$ and $\Sigma_{hu} =$

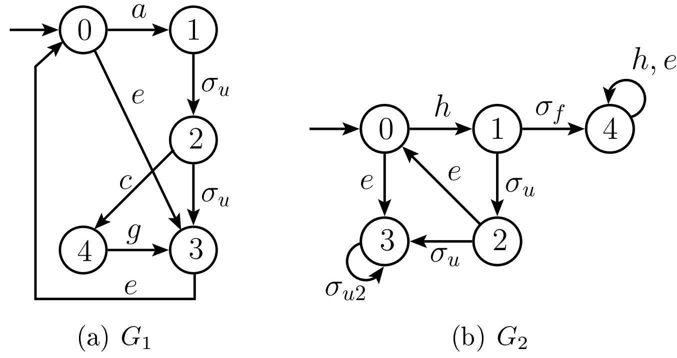


Figure 5.13: Automaton G_1 (a); and automaton G_2 (b) of Example 5.6.

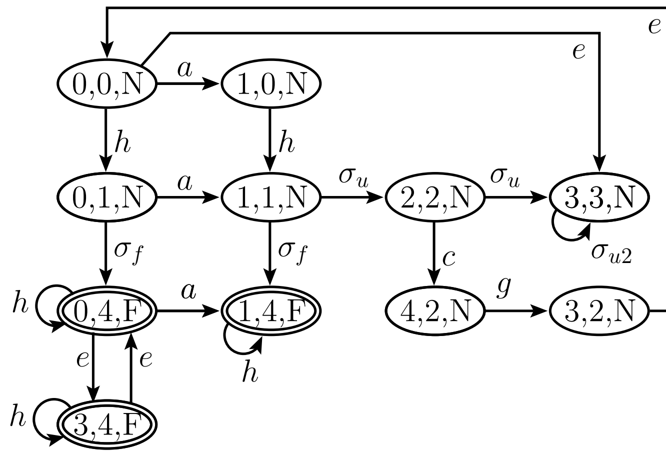


Figure 5.14: Automaton G_F of Example 5.6.

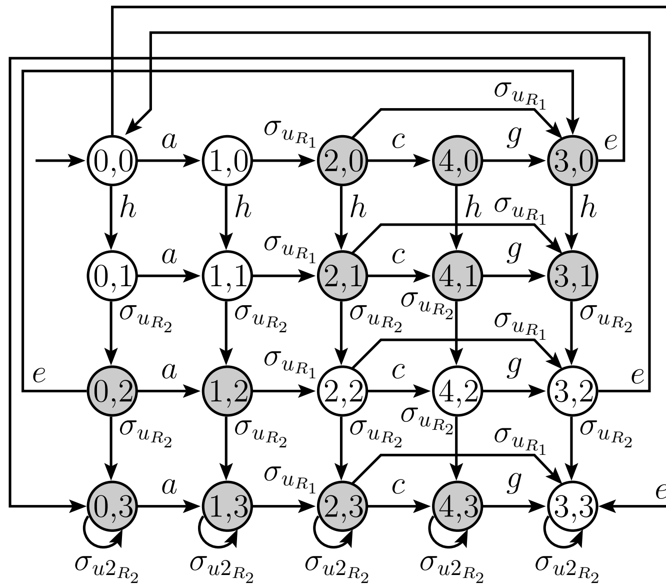


Figure 5.15: Automaton G_N^R of Example 5.6.

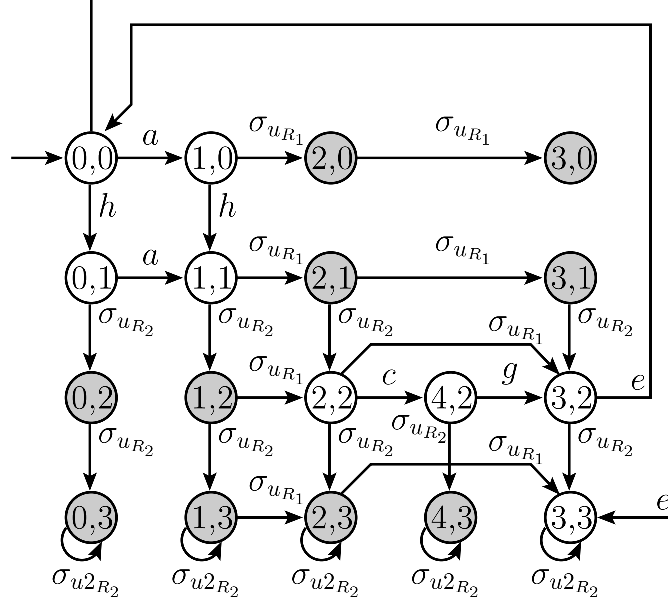


Figure 5.16: Automaton $G^R_{N,c}$ of Example 5.6.

$\{a_{pc}, a_{dp}, s_i, c_a, a_{pp}, a_{dc}, \sigma_f\}$, respectively, where $\Sigma_{cb,o} = \{a_1, a_2, cr_{on}, c_{off}, a_{dc}, cl_{on}, d_c\}$ and $\Sigma_{hu,o} = \{a_{dp}, s_i, c_a, a_{dc}\}$ are the sets of observable events of G_{cb} and G_{hu}, and $\Sigma_{cb,uo} = \{a_{pc}\}$ and $\Sigma_{hu,uo} = \{a_{pc}, a_{pp}, \sigma_f\}$ are the sets of unobservable events of G_{cb} and G_{hu}, respectively. The sets of events, observable events, and unobservable events of the plant are, respectively, $\Sigma_p = \Sigma_{cb} \cup \Sigma_{hu}$, $\Sigma_{p,o} = \Sigma_{cb,o} \cup \Sigma_{hu,o}$, and $\Sigma_{p,uo} = \Sigma_{cb,uo} \cup \Sigma_{hu,uo}$. In Section 3.5, we have presented how a synchronized Petri net diagnoser $\mathcal{N}_{D_p}$ is obtained for G_p and how the synchronous diagnosis is carried out. Now, let us consider the conditional synchronous diagnosis of G_p. Before the construction of the conditional synchronous Petri net diagnoser $\mathcal{N}_{D_p,c}$ for the system G_p, it is necessary to verify if the language generated by G_p, L_p, is conditionally synchronously diagnosable. Since L_p is synchronously diagnosable, then, according to Remark 5.2, L_p is also conditionally synchronously diagnosable, and the conditional synchronous Petri net diagnoser $\mathcal{N}_{D_p,c}$ can be constructed.

According to Algorithm 5.2, the conditional synchronous Petri net diagnoser $\mathcal{N}_{D_p,c}$ is computed and it is shown in Figure 5.17. It can be seen that in $\mathcal{N}_{D_p,c}$, in order to a transition to fire, it is necessary that the transition is enabled, the system executes the event that labels the transition and its associated condition is true. In Table 5.1 we show the meaning of the conditions associated to the transitions of $\mathcal{N}_{D_p,c}$, obtained from the nonfailure automaton behavior model of the system G_{N_p}. In order to simplify the notation, in Table 5.1, each place represent its marking. The conditions whose boolean value is always true are not represented in $\mathcal{N}_{D_p,c}$.

Let us now show how the conditional synchronous diagnosis is carried out by using $\mathcal{N}_{D_p,c}$. Suppose that the failure trace $s = a_1 cr_{on} c_{off} \sigma_f a_{dp} s_i a_{dp} c_a s_i a_{dc}$ has been

executed by the system. After the observation of trace $a_1 cr_{on} c_{off} \sigma_f a_{dp} s_i$, the places p_{C_3}, p_{C_4}, p_{H_3}, and p_{H_4} have tokens. When event a_{dc} is observed again, transitions $t_{2,6}$ and $t^c_{2,8}$ will fire, removing the tokens of places p_{H_3} and p_{H_4}, which enables transition t_{f_2}, that fires, diagnosing the occurrence of the failure event. Notice that, after the observation of trace $a_1 cr_{on} c_{off} \sigma_f a_{dp} s_i$ the condition $[\overline{C_5}] = [\overline{p_{C_8}}]$ of transition $t^c_{2,8}$ is true, since place p_{C_8} does not have a token. This happens because, in G_{N_p}, the unique state that has the second coordinate equal to H_4 and event a_{dp} is active is state (C_8, H_4) of G_{N_p}. Thus, after the observation of trace $a_1 cr_{on} c_{off} \sigma_f a_{dp} s_i$, only event a_2 is possible in the conditional nonfailure behavior of the system and, since event a_{dp} is executed, the failure is diagnosed.

It is important to notice that, in Section 3.5, the Petri net diagnoser $\mathcal{N}_{D_p}$ is computed and the same failure trace $s = a_1 cr_{on} c_{off} \sigma_f a_{dp} s_i a_{dp} c_a s_i a_{dc}$ is considered in order to illustrate the synchronous diagnosis using $\mathcal{N}_{D_p}$. In order to facilitate the comparison, in Figure 5.18 we show the Petri net diagnoser $\mathcal{N}_{D_p}$ computed in Section 3.5. If the system executes trace s, the failure event σ_f is diagnosed using diagnoser $\mathcal{N}_{D_p}$, only after the observation of event a_{dc}. Thus, for this system, considering the same failure trace s, by using diagnoser $\mathcal{N}_{D_p}$ it is necessary that the system executes three more events in order to $\mathcal{N}_{D_p}$ diagnose the failure event σ_f when compared to diagnoser $\mathcal{N}_{D_{p,c}}$. Moreover, for the same failure trace, in $\mathcal{N}_{D_p}$, the failure is diagnosed after the firing of transition t_{f_1} while in $\mathcal{N}_{D_{p,c}}$, the occurrence of σ_f is diagnosed after the firing of transition t_{f_2}. More details about the implementation of $\mathcal{N}_{D_{p,c}}$ can be found in MOTA VERAS [66].

The delay bound for conditional synchronous diagnosis of the system G_p can be computed according to Algorithm 3.3 using the verifier $G^{SD}_{V,c}$ instead of G^{SD}_V. The delay bound for conditional synchronous diagnosis is $z^* = 12$, which corresponds to the delay bound for monolithic diagnosis of the system G_p. The delay bound for synchronous diagnosis of system G_p, obtained in Section 3.5, is equal to 15. Notice that, by using the conditional synchronous diagnosis scheme, we have achieved, in this example, the same value for the delay bound than using the monolithic diagnosis approach.

5.4 Final remarks

In this chapter, we propose a refinement in the synchronous diagnosis scheme by adding conditions to the synchronized Petri net diagnoser transitions. These conditions are based on the nonfailure global behavior model of the system. We show that, with this refinement, the augmented nonfailure language of the system for synchronous diagnosis can be decreased, which implies that systems that are not synchronously diagnosable can be conditionally synchronously diagnosable, and that

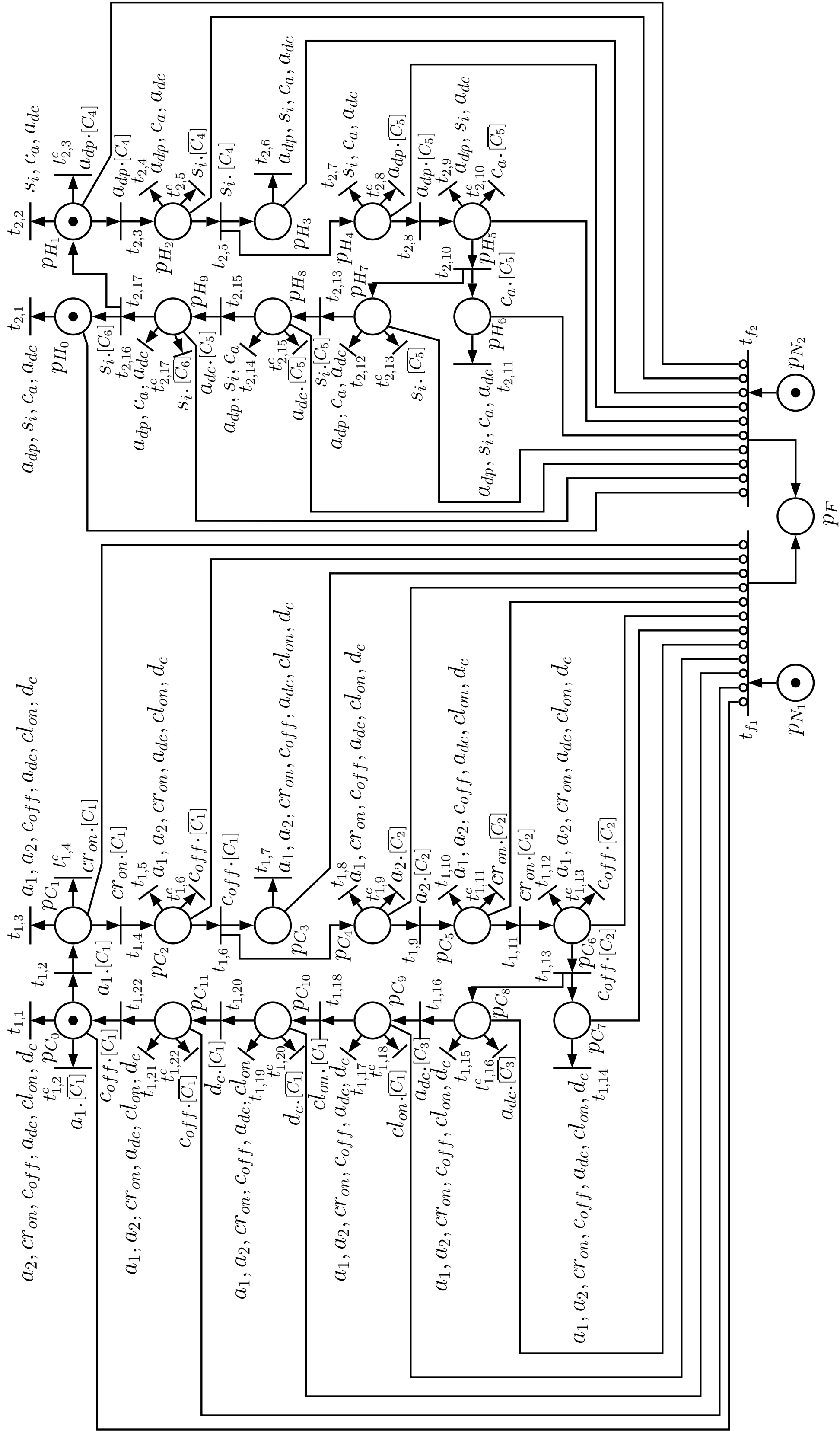


Figure 5.17: Conditional synchronized Petri net diagnoser $\mathcal{N}_{D_p,c}$.

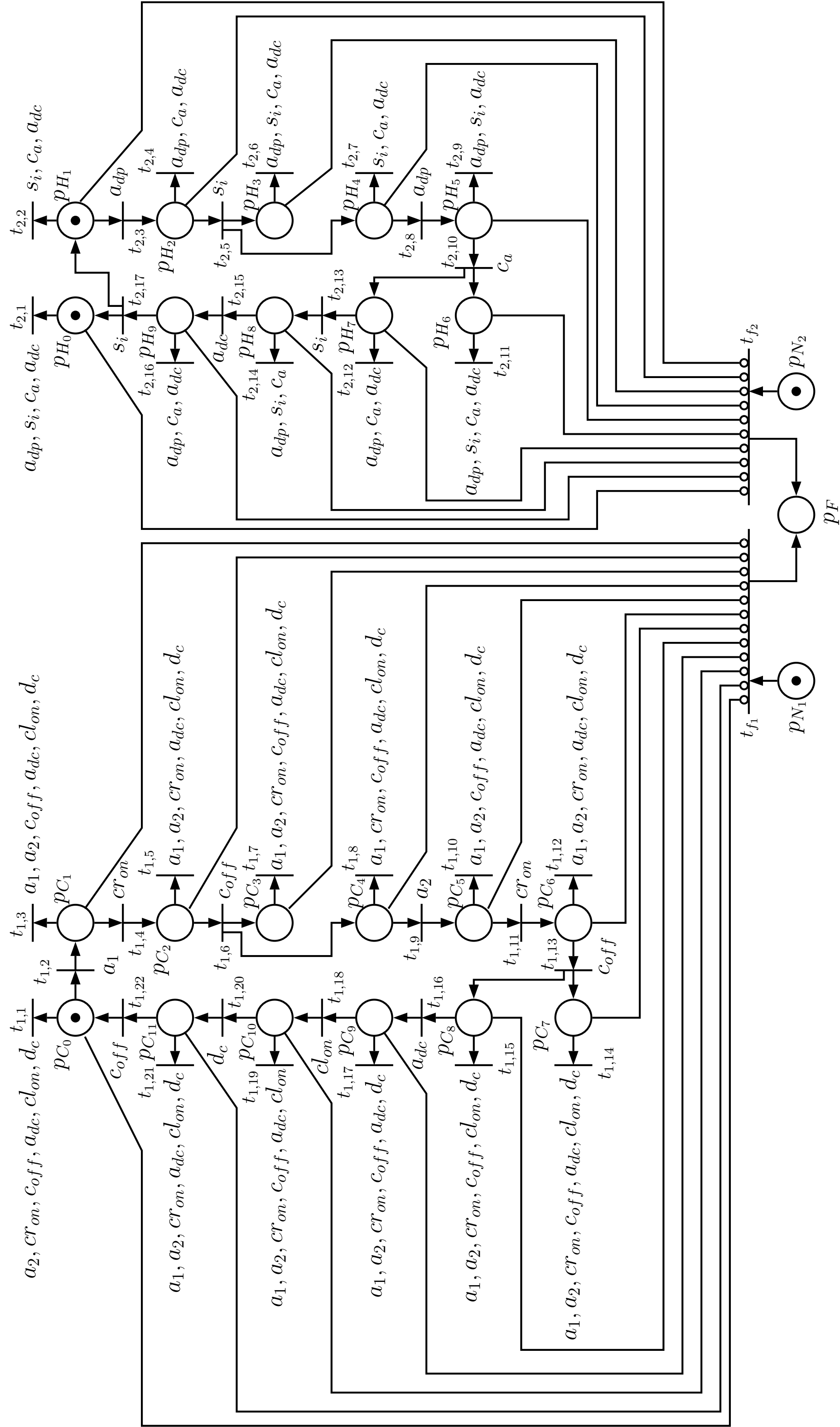


Figure 5.18: Petri net diagnoser $\mathcal{N}_{D_p}$.

Table 5.1: Conditions that label the transitions of the Petri net $\mathcal{N}_{D_p,c}$.

Condition	Meaning
$[C_1]$	$[p_{H_0}, p_{H_9}]$
$[C_2]$	$[p_{H_1}, p_{H_2}, p_{H_3}]$
$[C_3]$	$[p_{H_8}]$
$[C_4]$	$[p_{C_4}, p_{C_5}, p_{C_6}, p_{C_7}]$
$[C_5]$	$[p_{C_8}]$
$[C_6]$	$[p_{C_9}, p_{C_{10}}, p_{C_{11}}]$
$[\overline{C_1}]$	$[\overline{p_{H_0}}.\overline{p_{H_9}}]$
$[\overline{C_2}]$	$[\overline{p_{H_1}}.\overline{p_{H_2}}.\overline{p_{H_3}}]$
$[\overline{C_3}]$	$[\overline{p_{H_8}}]$
$[\overline{C_4}]$	$[\overline{p_{C_4}}.\overline{p_{C_5}}.\overline{p_{C_6}}.\overline{p_{C_7}}]$
$[\overline{C_5}]$	$[\overline{p_{C_8}}]$
$[\overline{C_6}]$	$[\overline{p_{C_9}}.\overline{p_{C_{10}}}.\overline{p_{C_{11}}}]$

even if the system is synchronously diagnosable, it is possible to improve the failure diagnosis by reducing the delay bound for diagnosis by adding the conditions in the Petri net diagnoser. A method to verify the conditional synchronous diagnosability, based on the synchronous diagnosability verifier presented in Chapter 3, is also presented.

Chapter 6

Conclusion and future research topics

In this work, a synchronous Petri net diagnoser (SPND) for discrete event systems modeled as automata is proposed. The SPND provides the state estimate of the nonfailure behavior of the component models of the system after the observation of a trace. In general, this state estimate constitutes a larger set than the state estimate of the nonfailure behavior of the composed system. Thus, the notion of synchronous diagnosability is presented and an algorithm to verify this property is proposed. We show that a system can be diagnosable and not synchronously diagnosable. Although the verifier automaton used to verify the synchronous diagnosability has exponential growth in the number of system components, it can be computed offline and, if the system is synchronously diagnosable, the SPND can be implemented. Since the construction of the global plant model for synchronous diagnosis is avoided, the SPND has polynomial computational complexity with the number of system components. Since the nonfailure language for synchronous diagnosis can be a larger set than the nonfailure language of the system, we also propose a method for the computation of the maximum delay bound for synchronous diagnosis based on the method proposed in TOMOLA *et al.* [77].

Moreover, we extend the notion of synchronous diagnosability to a decentralized setting using a scheme similar to the one presented in protocol 3 of DEBOUK *et al.* [17]. In order to do so, we implement a local Petri net diagnoser associated with each local component of the system. Since in this diagnosis scheme the observation is decentralized, a local observable event can be unobservable to another site. We have shown that, because of this fact, a system can be synchronously diagnosable and not synchronously codiagnosable. However, if a system is synchronously codiagnosable, it is synchronously diagnosable and, ultimately, diagnosable. We also show that the modular diagnosis scheme presented in CONTANT *et al.* [59] can be seen as a particular case of the synchronous decentralized diagnosis proposed in this work.

A modification of the SPND based on the nonfailure behavior model of the global system in order to refine the synchronous diagnosis is also proposed. This modifi-

cation relies on the addition of conditions to the synchronous Petri net diagnoser transitions. We show that such modification can decrease the observed nonfailure language for synchronous diagnosis, leading to the notion of conditional synchronous diagnosability. Since the nonfailure language considered for conditional synchronous diagnosis can be a smaller set that the nonfailure language for synchronous diagnosis, systems that are not synchronously diagnosable can be conditionally synchronously diagnosable. Moreover, for systems that are both synchronously diagnosable and conditionally synchronously diagnosable, the delay bound for the diagnosis of the failure event can be decreased when the conditional synchronous diagnosis scheme is used.

In the sequel, we summarize the main contributions of this work.

- A new failure diagnosis scheme based on the observation of the nonfailure models of the components of the system, called synchronous diagnosis, is proposed.
- The synchronous diagnosis does not use the global plant model for diagnosis, reducing the computational cost for diagnosis of DESs modeled as automata.
- The notion of synchronous diagnosability and a method for the verification of this property are proposed.
- An algorithm for the computation of the maximum delay bound for synchronous diagnosis is proposed.
- The synchronous diagnosis scheme is extended to a decentralized setting, leading to the notion of synchronous codiagnosability.
- A method for the verification of synchronous codiagnosability is presented.
- A comparison between modular diagnosis and decentralized synchronous diagnosis is presented, where we have shown that the modular diagnosis is a particular case of the synchronous decentralized diagnosis scheme.
- The synchronous diagnosis is refined, and the notion of conditional synchronous diagnosability is presented.
- A method for the verification of conditional synchronous diagnosability of DESs is proposed.
- Practical implementations of all diagnosis schemes presented in this work were carried out, validating the methods for the diagnosis of a manufacturing plant.

Since, in this work, a new diagnosis scheme for DESs is proposed, there are several research topics that can be developed based on the results presented in this thesis. In the sequel, we present possible research topics that can be carried out from this work.

(*i*) The conditional synchronous diagnosis scheme can be generalized to a distributed implementation. In this setting, local Petri net diagnosers can be constructed for each component of the system and be connected through a communication network. The observation of events and local state estimates can be used in order to refine the diagnosis decision based on the global non-failure behavior model of the system. In order to do so, different network architectures can be considered and communication protocols must be developed.

(*ii*) Depending on the system, the construction of local Petri nets associated with some components are not necessary for the synchronous diagnosis. The computational complexity of the synchronous diagnoser can be decreased by searching for the components of the system that are strictly necessary for the diagnosis of the failure event.

(*iii*) If different sets of components can be used for synchronous diagnosis, different criteria can be established in order to support the choice of which set of components is more appropriate for synchronous diagnosis or decentralized synchronous diagnosis.

All results presented in this thesis have been published or submitted for publication. In the sequel, we present the contributions related to this work.

(*i*) Online fault diagnosis of modular discrete-event systems [73].

(*ii*) Failure diagnosability of modular discrete-event systems [85].

(*iii*) Algorithms for the verification of synchronous diagnosability and computation of the delay bound for diagnosis of modular discrete event systems [86].

(*iv*) Conditional synchronized diagnoser for modular discrete-event systems [87].

(*v*) Synchronous Codiagnosability of Modular Discrete-Event Systems [83].

(*vi*) "Polynomial Time Verification of Decentralized Diagnosability of Discrete Event Systems" vs. "Decentralized Failure Diagnosis of Discrete Event Systems": A Critical Appraisal [27].

(*vii*) Robust Disjunctive-Codiagnosability of Discrete-Event Systems Against Permanent Loss of Observations [77].

(*viii*) Synchronous Diagnosis of Discrete-Event Systems - Submitted for publication [74].

(*ix*) Synchronous codiagnosability of discrete-event systems - Submitted for publication [84].

Bibliography

[1] CASSANDRAS, C., LAFORTUNE, S. *Introduction to Discrete Event System.* Secaucus, NJ, Springer-Verlag New York, Inc., 2008.

[2] HOPCROFT, J. E., MOTWANI, R., ULLMAN, J. D. *Introduction to automata theory, languages, and computation.* Boston, Addison Wesley, 2006.

[3] LAWSON, M. V. *Finite automata.* Florida, CRC Press, 2003.

[4] DAVID, R., ALLA, H. *Discrete, Continuous and Hybrid Petri Nets.* Springer, 2005.

[5] MIYAGI, P. E. *Controle programável: fundamentos do controle de sistemas a eventos discretos.* Edgard Blücher, 1996.

[6] RAMADGE, P. J., WONHAM, W. M. "The control of discrete event systems". In: *Proc. IEEE, Special Issue on Discrete Event Systems*, v. 77, pp. 81–98, 1989.

[7] THISTLE, J. G. "Supervisory control of discrete event systems", *Mathematical and Computer Modelling*, v. 23, n. 11-12, pp. 25–53, 1996.

[8] DE QUEIROZ, M. H., CURY, J. E. "Modular control of composed systems". In: *Proceedings of the American Control Conference*, v. 6, pp. 4051–4055, Chicago, IL, USA, 2000.

[9] DE QUEIROZ, M. H., CURY, J. E. "Modular supervisory control of large scale discrete event systems". In: *Boel R., Stremersch G. (eds) Discrete Event Systems. The Springer International Series in Engineering and Computer Science*, v. 569, Springer, Boston, MA, pp. 103–110, 2000.

[10] DE QUEIROZ, M. H., CURY, J. E. "Synthesis and implementation of local modular supervisory control for a manufacturing cell". In: *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*, pp. 377–382, Zaragoza, Spain, 2002.

[11] ÅKESSON, K., FLORDAL, H., FABIAN, M. "Exploiting modularity for synthesis and verification of supervisors". In: *15th IFAC triennial world congress*, v. 35, pp. 175–180, Barcelona, Spain, 2002. Elsevier.

[12] HILL, R. C., TILBURY, D. M. "Modular supervisory control of discrete-event systems with abstraction and incremental hierarchical construction". In: *Discrete Event Systems, 2006 8th International Workshop on*, pp. 399–406, 2006.

[13] PENA, P. N., CURY, J. E., LAFORTUNE, S. "Verification of nonconflict of supervisors using abstractions", *IEEE Transactions on Automatic Control*, v. 54, n. 12, pp. 2803–2815, 2009.

[14] SAMPATH, M., SENGUPTA, R., LAFORTUNE, S., et al. "Diagnosability of discrete-event systems", *IEEE Transactions on Automatic Control*, v. 40, n. 9, pp. 1555–1575, 1995.

[15] SAMPATH, M., SENGUPTA, R., LAFORTUNE, S., et al. "Failure diagnosis using discrete-event models", *IEEE Transactions on Control Systems Technology*, v. 4, n. 2, pp. 105–124, 1996.

[16] LIN, F. "Diagnosability of discrete event systems and its applications", *Journal of Discrete Event Dynamic Systems*, v. 4, n. 2, pp. 197–212, 1994.

[17] DEBOUK, R., LAFORTUNE, S., TENEKETZIS, D. "Coordinated decentralized protocols for failure diagnosis of discrete event systems", *Discrete Event Dynamic Systems: Theory and Applications*, v. 10, n. 1, pp. 33–86, 2000.

[18] SENGUPTA, R., TRIPAKIS, S. "Decentralized diagnosability of regular languages is undecidable". In: *Proceedings of the 41st IEEE Conference on Decision and Control (CDC)*, v. 1, pp. 423–428, 2002.

[19] QIU, W., KUMAR, R. "Decentralized failure diagnosis of discrete event systems", *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans*, v. 36, n. 2, pp. 384–395, 2006.

[20] WANG, Y., YOO, T.-S., LAFORTUNE, S. "Diagnosis of discrete event systems using decentralized architectures", *Discrete Event Dynamic Systems: Theory And Applications*, v. 17, pp. 233–263, 2007.

[21] CASSEZ, F. "A note on fault diagnosis algorithms". In: *Proceedings of the 48th IEEE Conference on Decision and Control held jointly with the 28th Chi-*

nese Control Conference, CDC/CCC., pp. 6941–6946, Shanghai, China, 2009.

[22] CARVALHO, L. K., MOREIRA, M. V., BASILIO, J. C. "Generalized robust diagnosability of discrete event systems". In: *18th IFAC World Congress*, pp. 8737–8742, Milano, Italy, 2011.

[23] CARVALHO, L. K., BASILIO, J. C., MOREIRA, M. V. "Robust diagnosis of discrete-event systems against intermittent loss of observations", *Automatica*, v. 48, n. 9, pp. 2068–2078, 2012.

[24] BASILIO, J. C., LIMA, S. T. S., LAFORTUNE, S., et al. "Computation of minimal event bases that ensure diagnosability", *Discrete Event Dynamic Systems: Theory And Applications*, v. 22, pp. 249–292, 2012.

[25] CARVALHO, L. K., MOREIRA, M. V., BASILIO, J. C., et al. "Robust diagnosis of discrete-event systems against permanent loss of observations", *Automatica*, v. 49, n. 1, pp. 223–231, 2013.

[26] CABRAL, F. G., MOREIRA, M. V., DIENE, O., et al. "A Petri net diagnoser for discrete event systems modeled by finite state automata", *IEEE Transactions on Automatic Control*, v. 60, n. 1, pp. 59–71, 2015.

[27] MOREIRA, M. V., BASILIO, J. C., CABRAL, F. G. ""Polynomial Time Verification of Decentralized Diagnosability of Discrete Event Systems" Versus "Decentralized Failure Diagnosis of Discrete Event Systems": A Critical Appraisal", *IEEE Transactions on Automatic Control*, v. 61, n. 1, pp. 178–181, 2016.

[28] SANTORO, L. P. M., MOREIRA, M. V., BASILIO, J. C. "Computation of minimal diagnosis bases of Discrete-Event Systems using verifiers", *Automatica*, v. 77, pp. 93–102, 2017.

[29] TRIPAKIS, S. "Fault diagnosis for timed automata". In: *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pp. 205–221. Springer, 2002.

[30] BOUYER, P., CHEVALIER, F., D'SOUZA, D. "Fault diagnosis using timed automata". In: *8th International Conference on Foundations of Software Science and Computation Structures*, pp. 219–233, Edinburgh, UK, 2005. Springer.

[31] ZAD, S. H., KWONG, R., WONHAM, W. "Fault diagnosis in discrete-event systems: Incorporating timing information", *IEEE Transactions on Automatic Control*, v. 50, n. 7, pp. 1010–1015, 2005.

[32] WU, Y., HADJICOSTIS, C. N. "Algebraic approaches for fault identification in discrete-event systems", *IEEE Transactions on Automatic Control*, v. 50, n. 12, pp. 2048–2055, 2005.

[33] GENC, S., LAFORTUNE, S. "Distributed diagnosis of place-bordered Petri nets", *IEEE Transactions on Automation Science and Engineering*, v. 4, n. 2, pp. 206–219, 2007.

[34] RAMIREZ-TREVINO, A., RUIZ-BELTRAN, E., RIVERA-RANGEL, I., et al. "Online fault diagnosis of discrete event systems. A Petri net-based approach", *IEEE Transactions on Automation Science and Engineering*, v. 4, n. 1, pp. 31–39, 2007.

[35] BASILE, F., CHIACCHIO, P., DE TOMMASI, G. "An efficient approach for online diagnosis of discrete event systems", *IEEE Transactions on Automatic Control*, v. 54, n. 4, pp. 748–759, 2009.

[36] CABASINO, M. P., GIUA, A., POCCI, M., et al. "Discrete event diagnosis using labeled Petri nets. An application to manufacturing systems", *Control Engineering Practice*, v. 19, n. 9, pp. 989–1001, 2011.

[37] FANTI, M. P., MANGINI, A. M., UKOVICH, W. "Fault detection by labeled Petri nets and time constraints". In: *3rd International Workshop on Dependable Control of Discrete Systems (DCDS)*, pp. 168–173, Saarbrucken, Germany, 2011.

[38] CABASINO, M., GIUA, A., LAFORTUNE, S., et al. "A New Approach for Diagnosability Analysis of Petri Nets using Verifiers Nets", *IEEE Transactions on Automatic Control*, v. 57, n. 12, pp. 3104–3117, 2012.

[39] FANTI, M. P., MANGINI, A. M., UKOVICH, W. "Fault detection by labeled Petri nets in centralized and distributed approaches", *IEEE Transactions on Automation Science and Engineering*, v. 10, n. 2, pp. 392–404, 2013.

[40] ZAYTOON, J., LAFORTUNE, S. "Overview of fault diagnosis methods for discrete event systems", *Annual Reviews in Control*, v. 37, n. 2, pp. 308–320, 2013.

[41] ZAD, S., KWONG, R., WONHAM, W. "Fault diagnosis in discrete-event systems: framework and model reduction", *IEEE Transactions on Automatic Control*, v. 48, n. 7, pp. 1199–1212, 2003.

[42] CLAVIJO, L. B., BASILIO, J. C. "Empirical studies in the size of diagnosers and verifiers for diagnosability analysis", *Discrete Event Dynamic Systems*, v. 27, n. 4, pp. 1–39, 2017.

[43] SCHMIDT, K. "Abstraction-based verification of codiagnosability for discrete event systems", *Automatica*, v. 46, n. 9, pp. 1489–1494, 2010.

[44] QIU, W., KUMAR, R. "Distributed diagnosis under bounded-delay communication of immediately forwarded local observations", *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, v. 38, n. 3, pp. 628–643, 2008.

[45] KEROGLOU, C., HADJICOSTIS, C. N. "Distributed diagnosis using predetermined synchronization strategies". In: *IEEE 53rd Annual Conference on Decision and Control (CDC)*, pp. 5955–5960, Los Angeles, CA, USA, 2014.

[46] YAMAMOTO, T., TAKAI, S. "Conjunctive decentralized diagnosis of discrete event systems", *IFAC Proceedings Volumes*, v. 46, n. 22, pp. 67–72, 2013.

[47] GIUA, A., SEATZU, C., CORONA, D. "Marking estimation of Petri nets with silent transitions", *IEEE Transactions on Automatic Control*, v. 52, n. 9, pp. 1695–1699, 2007.

[48] CABASINO, M. P., GIUA, A., SEATZU, C. "Fault detection for discrete event systems using Petri nets with unobservable transitions", *Automatica*, v. 46, pp. 1531–1539, 2010.

[49] CABASINO, M. P., GIUA, A., SEATZU, C. "Diagnosis using labeled Petri nets with silent or undistinguishable fault events", *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, v. 43, n. 2, pp. 345–355, 2013.

[50] CABASINO, M. P., GIUA, A., PAOLI, A., et al. "Decentralized Diagnosis of Discrete Event Systems using labeled Petri nets", *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, v. 43, n. 6, pp. 1477–1485, 2013.

[51] RAMIREZ-TREVINO, A., RUIZ-BELTRAN, E., RIVERA-RANGEL, I., et al. "Online fault diagnosis of discrete event systems. A Petri net-based ap-

proach", *IEEE Transactions on Automation Science and Engineering*, v. 4, n. 1, pp. 31–39, 2007.

[52] PANDALAI, D. N., HOLLOWAY, L. E. "Template languages for fault monitoring of timed discrete event processes", *IEEE Transactions on Automatic Control*, v. 45, n. 5, pp. 868–882, 2000.

[53] ROTH, M., LESAGE, J.-J., LITZ, L. "An FDI method for manufacturing systems based on an identified model". In: *13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM2009)*, pp. 1406–1411, Moscow, Russia, 2009.

[54] ROTH, M., LESAGE, J.-J., LITZ, L. "The concept of residuals for fault localization in discrete event systems", *Control Engineering Practice*, v. 19, n. 9, pp. 978–988, 2011.

[55] SAYED-MOUCHAWEH, M. "Decentralized fault free model approach for fault detection and isolation of discrete event systems", *European Journal of Control*, v. 18, n. 1, pp. 82–93, 2012.

[56] DEBOUK, R., MALIK, R., BRANDIN, B. "A modular architecture for diagnosis of discrete event systems". In: *41st IEEE Conference on Decision and Control*, pp. 417–422, Las Vegas, Nevada USA, 2002.

[57] PENCOLÉ, Y. "Diagnosability analysis of distributed discrete event systems". In: *Proceedings of the 16th European Conference on Artificial Intelligence*, pp. 38–42, Valencia, Spain, 2004.

[58] PENCOLÉ, Y., CORDIER, M.-O. "A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks", *Artificial Intelligence*, v. 164, n. 1-2, pp. 121–170, 2005.

[59] CONTANT, O., LAFORTUNE, S., TENEKETZIS, D. "Diagnosability of discrete event systems with modular structure", *Discrete Event Dynamic Systems: Theory And Applications*, v. 16, n. 1, pp. 9–37, 2006.

[60] ZHOU, C., KUMAR, R., SREENIVAS, R. S. "Decentralized modular diagnosis of concurrent discrete event systems". In: *9th Workshop on Discrete Event Systems*, pp. 388–393, Göteborg, Sweden, 2008.

[61] KAN JOHN, P., GRASTIEN, A., PENCOLÉ, Y. "Synthesis of a distributed and accurate diagnoser". In: *21st International Workshop on Principles of Diagnosis (DX-10)*, pp. 209–216, Portland, USA, 2010.

[62] SCHMIDT, K. W. "Verification of modular diagnosability with local specifications for discrete-event systems", *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, v. 43, n. 5, pp. 1130–1140, 2013.

[63] LI, B., BASILIO, J. C., KHLIF-BOUASSIDA, M., et al. "Polynomial Time Verification of Modular Diagnosability of Discrete Event Systems". In: *20th World Congress of the International Federation of Automatic Control*, pp. 14182–14187, Toulouse, France, 2017.

[64] GARCÍA, E., CORRECHER, A., MORANT, F., et al. "Centralized Modular Diagnosis and the Phenomenon of Coupling", *Discrete Event Dynamic Systems*, v. 16, n. 3, pp. 311–326, 2006.

[65] LUCIO, M. L. *Diagnóstico de falhas sincronizado de uma planta de manufatura.* Projeto de Graduação, Escola Politécnica - UFRJ, Rio de Janeiro, RJ, Brasil, 2015.

[66] MOTA VERAS, M. Z. *Diagnóstico de falhas em uma planta modular de montagem de cubos.* Projeto de Graduação, Escola Politécnica - UFRJ, Rio de Janeiro, RJ, Brasil, 2016.

[67] BASILIO, J. C., CARVALHO, L. K., MOREIRA, M. V. "Diagnose de falhas em sistemas a eventos discretos modelados por autômatos finitos", *Revista Controle & Automação*, v. 21, n. 5, pp. 510–533, 2010.

[68] ALAYAN, H., NEWCOMB, R. W. "Binary Petri-Net Relationships", *IEEE Transactions on Circuits and Systems*, v. 34, n. 5, pp. 565–568, 1987.

[69] JIANG, S., HUANG, Z., CHANDRA, V., et al. "A polynomial algorithm for testing diagnosability of discrete-event systems", *IEEE Transactions on Automatic Control*, v. 46, n. 8, pp. 1318–1321, 2001.

[70] YOO, T.-S., LAFORTUNE, S. "Polynomial-time verification of diagnosability of partially observed discrete-event systems", *IEEE Transactions on Automatic Control*, v. 47, n. 9, pp. 1491–1495, 2002.

[71] MOREIRA, M. V., JESUS, T. C., BASILIO, J. C. "Polynomial time verification of decentralized diagnosability of discrete event systems", *IEEE Transactions on Automatic Control*, v. 56, n. 7, pp. 1679–1684, 2011.

[72] BASILIO, J. C., LAFORTUNE, S. "Robust codiagnosability of discrete event systems". In: *American Control Conference (ACC)*, pp. 2202–2209, St. Louis, MO, USA, 2009.

[73] CABRAL, F. G., MOREIRA, M. V., DIENE, O. "Online fault diagnosis of modular discrete-event systems". In: *IEEE 54th Annual Conference on Decision and Control (CDC)*, pp. 4450–4455, Osaka, Japan, 2015.

[74] CABRAL, F. G., MOREIRA, M. V. "Synchronous Diagnosis of Discrete-Event Systems", *IEEE Transactions on Automatic Control*, 2017. Submitted for publication.

[75] LEI, F. *Computationally Efficient Supervisor Design for Discrete-Event Systems*. Ph.D. dissertation, University of Toronto, Toronto, Canada, 2007.

[76] WONHAM, W. "Notes on control of discrete event systems", 09 2017.

[77] TOMOLA, J. H. A., CABRAL, F. G., CARVALHO, L. K., et al. "Robust Disjunctive-Codiagnosability of Discrete-Event Systems Against Permanent Loss of Observations", *IEEE Transactions on Automatic Control*, v. 62, n. 11, pp. 5808–5815, 2017.

[78] CARVALHO, L. K., MOREIRA, M. V., C, B. J. "Diagnosability of intermittent sensor faults in discrete event systems", *Automatica*, v. 79, pp. 315–325, 2017.

[79] DASGUPTA, S., PAPADIMITRIOU, C., VAZIRANI, U. *Algorithms*. McGraw-Hill, 2008.

[80] YOO, T.-S., GARCIA, H. "Computation of fault detection delay in discrete-event systems". In: *Proceedings of the 14th International Workshop on Principles of Diagnosis, DX'03*, pp. 207–212, Washington, USA, 2003.

[81] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., et al. *Introduction to algorithms*. Massachusetts, MIT Press, 2007.

[82] "Christiani Sharpline - Automation Technology". 2015. Available in: <`http://www.cstt.in/AutomationTechnology.html`>.

[83] CABRAL, F. G., MOREIRA, M. V. "Synchronous Decentralized Diagnosis of Discrete-Event Systems". In: *20th World Congress of the International Federation of Automatic Control*, pp. 7025–7030, Toulouse, France, 2017.

[84] CABRAL, F. G., MOREIRA, M. V. "Synchronous Codiagnosability of Discrete-Event Systems", *Automatica*, 2017. Submitted for publication.

[85] CABRAL, F. G., MOREIRA, M. V., DIENE, O. "Diagnosticabilidade de falhas em sistemas a eventos discretos modulares". In: *Anais do XII Simpósio Brasileiro de Automação Inteligente - SBAI*, Natal, Brasil, 2015.

[86] CABRAL, F. G., TOMOLA, J. H., MOREIRA, M. V. "Algorithms for the verification of synchronous diagnosability and computation of the delay bound for diagnosis of modular discrete event systems". In: *Anais do XXI Congresso Brasileiro de Automática - CBA*, Espírito Santo, Brasil, 2016.

[87] CABRAL, F. G., VERAS, M. Z. M., MOREIRA, M. V. "Conditional Synchronized Diagnoser for Modular Discrete-Event Systems". In: *14th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, v. 2, pp. 88–97, Madrid, Spain, 2017.